Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

DESARROLLO DE UNA TOOLBOX PARA MEDIR LA RESPUESTA POSITIVA DE UN SISTEMA HACIA ESTÍMULOS EXTERNOS

Autora: Jessica López-Hazas Sacristán
Tutor: Francisco de Borja Rodríguez Ortiz

Julio 2017

DESARROLLO DE UNA TOOLBOX PARA MEDIR LA RESPUESTA POSITIVA DE UN SISTEMA HACIA ESTÍMULOS EXTERNOS

Autora: Jessica López-Hazas Sacristán
Tutor: Francisco de Borja Rodríguez Ortiz

Grupo de Neurocomputación Biológica (GNB)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2017

Resumen

Resumen

El objetivo de este proyecto es desarrollar una metodología que permita la detección de respuestas de un sistema ante un estímulo específico en condiciones de incertidumbre. En muchos casos, la actividad continua inherente a un sistema, la reverberación de la información, la latencia en el tiempo de respuesta y otros problemas complican la detección de respuestas con fiabilidad. Este es el caso de los experimentos llevados a cabo en neurociencia sobre las neuronas que componen el sistema nervioso.

En el campo de la neurociencia, no existe un consenso sobre qué protocolos deben seguirse a la hora de analizar los datos procedentes de mediciones sobre neuronas y, como consecuencia de esto, no siempre se aplican métodos robustos y estandarizados de detección de respuestas, así como tampoco existen formatos universales de representación y etiquetado de experimentos.

Teniendo en cuenta lo anterior, en este proyecto se implementa una metodología robusta de detección de respuestas basada en un método bayesiano propuesto por el GNB de la UAM y se desarrollan las herramientas software necesarias para que pueda aplicarse de forma fácil y automática a datos procedentes de diferentes experimentos, definiendo para ello un formato estándar de representación de los mismos.

La metodología implementada es validada mediante su aplicación a datos generados de forma artificial mediante un modelo matématico sencillo para después ser aplicado sobre datos procedentes de experimentos reales llevados a cabo sobre las neuronas del lóbulo antenal y el cuerpo fungiforme del sistema olfativo de la langosta con diferentes estímulos. Los resultados son analizados y las conclusiones extraídas gracias a la aplicación de la metodología son coherentes con lo que se sabe sobre el funcionamiento de dicho sistema.

Palabras Clave

Detección de respuestas, CPA, neurociencia, neurociencia computacional, sistema olfativo, disparos neuronales, inhibición, excitación, sensitividad, test de hipótesis, Bayes, células de Kenyon, neuronas de proyección

Abstract

The main goal of this project is to develope a methodology which allows the detection of responses to a specific stimulus in a system under uncertainty. Most of the times, the continous inherent activy of a system, information reverberation, latency in response time and other problems make it difficult to detect responses with reliability. This is the case of the experiments carried out in Neuroscience on neurons that compose the nervous system.

In Neuroscience, there is no consensus about which protocols must be followed during the analysis of the data collected from measurements on neurons and, as a result of this, robust and standardized methods for response detection are not always applied, nor are there universal formats for the representation and labeled of experiments.

Given the above, this project implements a robust methodology for response detection based on a bayesian method proposed by the GNB and developes the software tool necessary to apply it easily and automatically to analyze data from different experiments, defining for this purpose a standard format of representation.

The methodology developed is then validated through applying it to data that is artificially generated by a simple mathematical model. Then, it is applied to real data from experiments carried out on neurons located in the antennal lobe and the mushroom body in the locust olfactory system. The results obtained are analyzed and the conclusions drawn from them are consistent with what it is known about how this system works.

Key words

Response detection, CPA, Bayes, Neuroscience, Computational Neuroscience, olfactory system, neural spikes, inhibition, excitation, sensitivity, hypothesis test, Kenyon cells, Projection neurons

Agradecimientos

En primer lugar, me gustaría mostrar mi agradecimiento a Francisco B. Rodríguez por orientarme, no solo para la realización de este TFG, sino también para continuar estudiando este campo en el futuro.

Por otro lado, gracias a mi familia, en especial a mis padres, y a mi amiga Estela, quienes siempre me han dado todo su apoyo.

Índice general

Índice de Figuras IX

Índice de Tablas XI

1. Introducción 1

1.1. Motivación . . . 1

1.2. Objetivos y enfoque . . . 1

1.3. Estructura de la memoria . . . 2

2. Estado del arte 5

2.1. Introducción . . . 5

2.2. Neuronas y disparos neuronales . . . 5

2.3. Sistema olfativo de la langosta voladora . . . 6

2.4. Técnicas de Change Point Analysis (CPA) . . . 8

2.4.1. Planteamiento del problema . . . 8

2.4.2. Técnicas de CPA paramétricas . . . 9

2.4.3. Técnicas de CPA no paramétricas . . . 9

3. Diseño y desarrollo 11

3.1. Método bayesiano de CPA . . . 11

3.2. Modelo simple de generación de eventos . . . 12

3.3. Requisitos del programa . . . 15

3.4. Diseño del programa . . . 16

3.4.1. Diagrama de clases . . . 16

3.4.2. Formato y representación de los datos . . . 18

3.4.3. Implementación del análisis de datos . . . 18

4. Experimentos y análisis de resultados 23

4.1. Validación de la funcionalidad mediante datos generados por el modelo . . . 23

4.1.1. Estimación del tamaño de intervalo óptimo para la estimación de la distribución de probabilidad . . . 23

4.1.2. Validación de la estimación del límite inferior de probabilidad de respuesta por medio del método Bayesiano 26
4.1.3. Validación de reducción de falsos positivos 28
4.2. Aplicación del método a datos reales de células KCs y PNs 29
4.2.1. Descripción de los conjuntos de datos 29
4.2.2. Aspectos generales 30
4.2.3. Sensitividad de las KCs vs. PNs 32
4.2.4. Codificación temporal 34

5. Conclusiones y trabajo futuro **37**
5.1. Conclusiones 37
5.2. Trabajo futuro 37

Glosario **39**

Bibliografía **40**

A. Figuras adicionales **43**

B. Herramientas software desarrolladas **51**
B.1. Formato de los datos 51
B.2. Descripción de las herramientas desarrolladas 53
B.3. Código 55
B.3.1. DataLoader 55
B.3.2. Experiment 59
B.3.3. ExperimentSet 59
B.3.4. DataAnalyzer 60
B.3.5. Sensitivity 65
B.3.6. BayesianTest 69
B.3.7. BayesianTestBootstrap 73
B.3.8. GaussianMethod 75
B.3.9. Plotter 79

Índice de Figuras

2.1. Neurona 5

2.2. Potencial en la membrana frente al tiempo durante un disparo neuronal 6

2.3. Distribución de disparos neuronales en un experimento de 20s de duración con 10 pruebas (eje Y). La neurona es estimulada entre los segundos 4 y 5 (líneas verticales). 7

2.4. Sistema olfativo de un insecto 7

3.1. Protocolo seguido en la realización de los experimentos 11

3.2. Distribución de disparos neuronales en un experimento de 20s de duración con 10 pruebas (eje Y). La neurona es estimulada entre los segundos 4 y 5. El estímulo utilizado tiene unos parámetros $\alpha = 0{,}65$ y $k = 8$. 13

3.3. PSTH de un experimento de 20s de duración con 10 pruebas. La neurona es estimulada entre los segundos 4 y 5. El estímulo utilizado tiene unos parámetros $\alpha = 0{,}65$ y $k = 8$. 14

3.4. PSTH de un experimento de 20s de duración con 10 pruebas. La neurona es estimulada entre los segundos 4 y 5. El estímulo utilizado tiene unos parámetros $\alpha = -0{,}80$ y $k = 2$. 14

3.5. PSTH de un experimento de 20s de duración con 10 pruebas. La neurona es estimulada entre los segundos 4 y 5. El estímulo utilizado tiene unos parámetros $\alpha = -0{,}80$ y $k = 2$. 15

3.6. Diagrama de clases del programa de herramientas implementado 17

4.1. Datasets generados para la validación 24

4.2. $C(\Delta)$ para el dataset 1 y el dataset 2 25

4.3. PSTHs de los datasets con los valores de Δ seleccionados 26

4.4. Distribución de eventos, función de coste y PSTH con $\Delta = 1{,}8$ para el dataset 3 . 27

4.5. PSTHs de los datasets generados para la validación del método Bayesiano 27

4.6. Porcentaje de falsos positivos en función de p_r 29

4.7. Superficies del límite inferior de probabilidad de respuesta en función de diversos tamaños y puntos de inicio de la ventana de estimulación para dos experimento sobre KC's y dos experimentos sobre PN's 31

4.8. Diagramas de activación para KC's y PN's. El color negro indica que la neurona responde a un olor específico mientras el blanco corresponde a la ausencia de respuesta. 32

4.9. Sensitividad en el intervalo de estimulación para KC's y PN's 33

4.10. Distribuciones del límite inferior de probabilidad de respuesta para KC's y PN's . 33

4.11. Disparos neuronales para los odorantes hexanol y geraniol a concentraciones mínima (0.001) y máxima (1) 35

4.12. Disparos neuronales para los odorantes geraniol, hexanol y octanol a concentración máxima 36

A.1. Límites inferiores de probabilidad de respuesta para los datasets D_1, D_2 y D_3 en función del punto de inicio y duración de la ventana 44

A.2. Límites inferiores de probabilidad de respuesta para los datasets D_1, D_2 y D_3 en función del punto de inicio y duración de la ventana dentro del periodo de baseline 45

A.3. Diagramas de disparos neuronales y PSTHs para cuatro experimentos realizados sobre células KC's y PN's 46

A.4. $C(\Delta)$ suavizado para los experimentos seleccionados de KC's y PN's 47

A.5. Evolución de la sensitividad de las neuronas KC's en el tiempo 48

A.6. Evolución de la sensitividad de las neuronas PN's en el tiempo 49

Índice de Tablas

4.1. Características de los datos 30

1 Introducción

En esta sección se realiza una introducción al proyecto, explicando su motivación y enumerando los objetivos que se quieren conseguir. También se describe brevemente la estructura de este documento y el contenido de cada uno de sus capítulos.

1.1. Motivación

En el campo de la neurociencia, una metodología habitual para intentar medir la sensibilidad de un sistema ante una serie de estímulos externos es estimular el sistema durante determinado periodo de tiempo y registrar su actividad. A la hora de analizar los datos obtenidos de este modo, es de crucial importancia poder determinar con cierto grado de certeza si se ha producido una respuesta ante los estímulos o no. La actividad derivada del estado interno del sistema, las posibles respuestas a otros estímulos distintos a los introducidos en el experimento o la reverberancia de información hacen complicada la detección de respuestas de forma fiable.

Para detectar respuestas neuronales ante un estímulo, es necesario examinar cómo evoluciona el número de disparos neuronales en el tiempo para así poder detectar cambios o anomalías durante la fase de estimulación que puedan ser evidencia de la existencia de una respuesta. En estadística, la detección de cambios en una distribución de eventos en una línea temporal es conocida como *Change Point Analysis* (CPA). Dada la naturaleza del problema, la aplicación de técnicas de CPA a datos procedentes de neuronas es muy adecuada. A pesar de ello, estas técnicas no siempre se emplean debido a la ausencia de métodos generales y robustos que permitan aplicarlos y a la falta de consenso sobre su utilización [1].

En este contexto, el Grupo de Neurocomputación Biológica (GNB) ha desarrollado un método de CPA basado en Bayes para poder determinar si un sistema responde ante un estímulo con cierto grado de certidumbre [2]. Este Trabajo Fin de Grado se centra en realizar una implementación de este método de forma que pueda ser utilizado fácilmente y de forma general para procesar los datos de experimentos sobre diferentes sistemas. A modo de validación, se aplicará a un conjunto de datos de experimentos sobre el sistema olfativo de la langosta a modo de validación.

1.2. Objetivos y enfoque

Este proyecto se centra en el desarrollo de una metodología de detección de respuestas ante estímulos propuesto por el GNB de la UAM y su aplicación a datos procedentes de experimentos sobre sensibilidad neuronal del sistema olfativo de la langosta. Para ello, en primer lugar se analizará el problema y el método propuesto, se diseñarán las utilidades necesarias, se implementarán y por último se validarán aplicándolo a los datos. Los objetivos del proyecto son los siguientes:

- Comprender el problema de la detección de respuesta ante estímulos cuando hay incertidumbre, el método bayesiano propuesto y otros métodos de CPA.
- Definir un formato unificado para la representación y etiquetado de los datos procedentes de diferentes experimentos y pruebas.
- Realizar estimaciones de la distribución de probabilidad seguida por los eventos registrados en el sistema.
- Determinar la probabilidad de respuesta del sistema ante un estímulo en diferentes intervalos de tiempo del experimento mediante el método bayesiano.
- Desarrollar una serie de herramientas que permitan la representación de aspectos relevantes del análisis de los datos.
- Realizar un análisis de datos reales procedentes del sistema olfativo de la langosta mediante las herramientas desarrolladas a modo de validación.
- Realizar una comparativa entre diferentes métodos.

En cuanto al lenguaje de programación en el que se implementarán las herramientas, se ha seleccionado Python ya que cuenta con numerosas bibliotecas y facilidades para procesar, analizar y representar datos, tales como Numpy, Scipy o Mathplotlib.

1.3. Estructura de la memoria

Esta memoria se organiza en los siguientes capítulos:

- **Estado del arte:** en esta sección se exponen los conceptos previos necesarios para la comprensión del trabajo realizado, tales como información relativa a las neuronas y cómo se producen los disparos neuronales, cómo se organiza y funciona el sistema olfativo de la langosta del que se han extraído los datos reales para la validación de la metodología presentada, y las técnicas de CPA más empleadas.
- **Diseño y desarrollo:** se exponen los pasos seguidos para el diseño y el desarrollo de la metodología presentada en este trabajo, presentando en primer lugar los fundamentos teóricos del método de CPA propuesto, los requisitos de las herramientas de software desarrolladas y cómo se organizan y, finalmente, detalles sobre la implementación de los puntos más interesantes de la metodología.
- **Experimentos y análisis de resultados**: en este capítulo se lleva a cabo la validación de la metodología desarrollada a través de su aplicación tanto a datos generados artificialmente como a datos reales medidos a través de experimentos sobre el sistema olfativo de la langosta. Sobre estos últimos, se presenta un análisis de los resultados que permite extraer conclusiones sobre su funcionamiento.

- **Conclusiones y trabajo futuro**: se exponen las conclusiones alcanzadas con este proyecto y un breve análisis final del mismo. Además, se explica cómo este trabajo va a seguir desarrollándose en el futuro.
- **Anexos:** figuras adicionales e información sobre las herramientas software desarrolladas y las clases y funciones que las componen.

2

Estado del arte

2.1. Introducción

En esta sección se exponen los conceptos y conocimientos previos necesarios para comprender este proyecto (disparos neuronales, sistema olfativo de la langosta), así como las principales técnicas que existen actualmente para la detección de respuestas en series temporales y el método propuesto por el GNB.

2.2. Neuronas y disparos neuronales

El sistema nervioso está compuesto principalmente por neuronas y células de Glía. Las neuronas son las unidades fundamentales para el procesamiento de la información en el sistema nervioso y su misión es recoger, integrar y propagar los impulsos nerviosos. Además del cuerpo celular donde se encuentran los orgánulos de la célula, poseen dos tipos de ramificaciones: dendritas, encargadas de recoger los impulsos procedentes de otras neuronas a las que está conectada, y los axones, que integran y propagan a otras neuronas los impulsos nerviosos [3, 4].

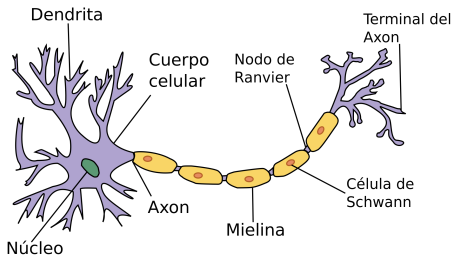


Figura 2.1: Neurona

Las neuronas, como el resto de células, están aisladas del exterior por una membrana citoplasmática. La diferencia de iones entre el interior y el exterior de la membrana hace que exista una diferencia de potencial, denominada potencial de reposo de la membrana, que suele encontrarse entre los -45mV y los -75mV, dependiendo del tipo de neurona, y teniendo en cuenta la convención de que el interior de la célula es negativo respecto al exterior. A partir de

variaciones en esta diferencia de potencial es como los impulsos nerviosos se transmiten por el sistema y, por ende, la información. Para ello, la membrana citoplasmática cuenta con canales iónicos que permiten o restringen la entrada de iones de potasio, sodio y cloro, principalmente, de forma que los cambios en el flujo de estos iones hiperpolarizan o despolarizan el interior de la célula. Algunos de estos canales son pasivos y permacecen siempre abiertos, mientras que otros modifican su permeabilidad debido a la acción de neurotransmisores, a cambios de potencial o a estímulos físicos.

Cuando una neurona es estimulada con un voltaje que supera su umbral de disparo, genera un potencial de acción que se propaga por el axón hasta sus terminales. Este potencial de acción se inicia por la despolarización de la membrana, al volverse esta más permeable a iones de sodio, que entran por los canales iónicos. La membrana se despolariza rápidamente hasta unos +30mV y después decae por la apertura de los canales de potasio y el cierre de los de sodio, hasta que la membrana queda hiperpolarizada para después retornar a su potencial de reposo. [3, 4] En la Fig. 2.2, puede verse representada la diferencia de potencial respecto al tiempo en el proceso de disparo de una neurona:

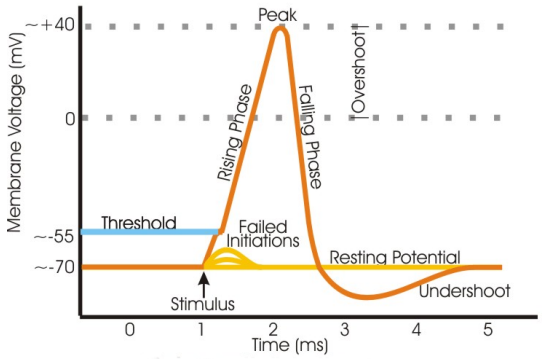


Figura 2.2: Potencial en la membrana frente al tiempo durante un disparo neuronal

Los datos que se analizarán en este proyecto consisten en series de disparos neuronales como el de la figura anterior medidos en diferentes neuronas del sistema olfativo de la langosta [5]. La inspección de los datos sobre los disparos neuronales es fundamental para poder establecer conclusiones sobre cómo se trata la información en el sistema nervioso, ya que se considera que los disparos neuronales y la forma en que estos se distribuyen codifican la información y su procesamiento [6]. Por tanto, los disparos neuronales constituirán los eventos de nuestro sistema, y su distribución será analizada para determinar si se corresponden o no a una respuesta neuronal frente a un estímulo externo. El formato de los datos consistirá en los momentos del tiempo en que se han detectado disparos neuronales a lo largo de las diferentes pruebas de un experimento. En la Fig. 2.3 se muestra un ejemplo de distribución de los disparos en el tiempo para un experimento consistente en diez pruebas con un mismo estímulo.

Como puede observarse en ella, la existencia constante de actividad neuronal inherente al sistema hace difícil determinar si los disparos registrados durante el periodo de estimulación o en mitad o después de este debido a la posible latencia, se corresponden con una respuesta ante el estímulo introducido o simplemente forman parte de esta actividad continua. En secciones siguientes se detallarán los métodos mediante los cuales la distribución de los disparos puede analizarse para resolver este problema.

2.3. Sistema olfativo de la langosta voladora

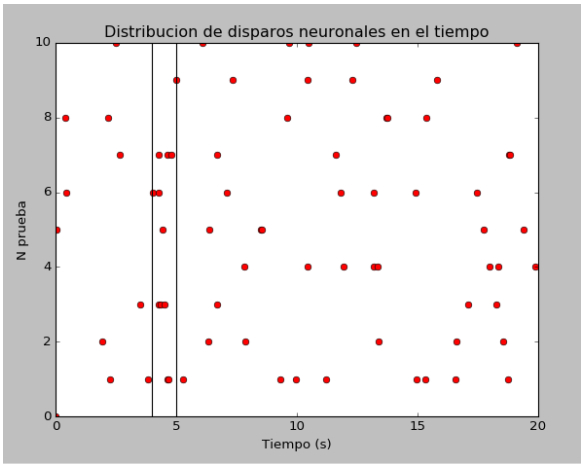


Figura 2.3: Distribución de disparos neuronales en un experimento de 20s de duración con 10 pruebas (eje Y). La neurona es estimulada entre los segundos 4 y 5 (líneas verticales).

Para poder comprender mejor los experimentos y datos que se van a analizar en este trabajo, en los siguientes párrafos se describe el sistema olfativo de la langosta y cómo se captura, transmite y procesa la información de los olores a través de las diferentes capas del sistema.

En primer lugar, los olores son captados por unas 90 000 neuronas receptoras olfativas en la antena del insecto, que codifican la información de los olores en un espacio de numerosas dimensiones. La información procedente de la antena pasa a la siguiente capa, el lóbulo antenal (LA), que contiene unas 1130 neuronas de dos tipos: neuronas locales (LNs) y neuronas de proyección (PNs). Las neuronas locales son 300 y pueden ser tanto inhibitorias como excitatorias. Su actividad repercute sobre las 830 neuronas de proyección (PNs), que son excitatorias. Estas neuronas de proyección conectan el LA con el cuerpo fungiforme (CF), el cual contiene unas 50 000 neuronas denominadas células de Kenyon (KCs), que proyectan la información recibida desde el LA en un espacio mucho mayor. Las PNs están directamente conectadas a las KCs mediante sus dendritas, de manera que cada PN está conectada a unas 600 KCs [5]. Este trabajo se centra en estudiar la respuesta que las KCs y las PNs muestran ante diferentes estímulos olfativos. Como

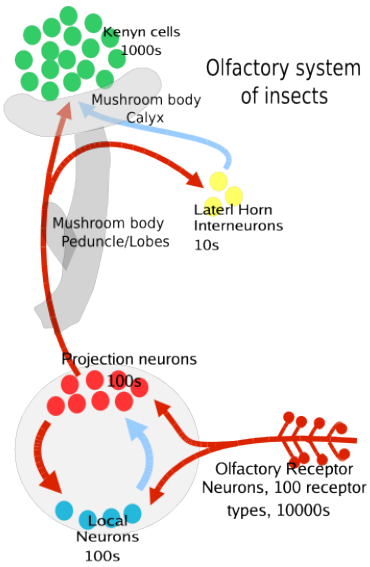


Figura 2.4: Sistema olfativo de un insecto

puede verse en la Fig. 2.4, la información captada en las antenas se comprime al pasar al lóbulo

antenal, lo que podría servir para captar las características generales de los olores detectados, para luego proyectarse en un espacio de más de dimensiones en el CF, de manera que todos los posibles olores detectados puedan codificarse de forma específica sin que se produzcan colisiones.

En [5], se llevó a cabo un estudio comparativo del comportamiento de la respuesta de las PNs y de las KCs midiendo los disparos neuronales en diferentes neuronas de cada capa para un conjunto de estímulos olfativos y en estado de reposo. A continuación se enumeran las características más importantes del comportamiento de las KCs, obtenidas a partir de las conclusiones alcanzadas en los experimentos anteriores:

- En baseline (periodos en los que el sistema no se expone a ningún estímulo), las KCs muestran una tasa media de disparos muy baja, unas 1000 veces inferior a las PNs.
- Las KCs rara vez responden ante estímulos olfativos y, cuando lo hacen, las respuestas son simples y breves en el tiempo. Por tanto, sus probabilidades de respuesta ante estímulos son muy bajas. En contraste, las PNs muestran una probabilidad de respuesta mucho mayor y exhiben patrones de respuesta complejos y cambiantes en el tiempo con respecto a los estímulos, lo que podría indicar que en ellos se emplea alguna codificación temporal para transmitir la información.

Los datos obtenidos en estos experimentos para las KCs y PNs serán los utilizados para la validación del método desarrollado en este trabajo.

2.4. Técnicas de Change Point Analysis (CPA)

2.4.1. Planteamiento del problema

Las técnicas de CPA se encargan de detectar si existen puntos en los que la distribución de una variable cambie a lo largo del tiempo y, de existir dichos puntos, se encarga de realizar una estimación de su posición en la línea temporal. De esta manera, el problema puede expresarse como una elección entre dos hipótesis. Si tenemos una variable aleatoria X que varía en función del tiempo, siendo X_t el valor de X en el instante de tiempo t, y suponemos que hay un solo *change point*, en el instante $t = \tau$, las dos hipótesis a contrastar serían las siguientes:

$$\begin{aligned} &\mathrm{H}_0 : X_t \sim F_0, \forall t \in T \\ &\mathrm{H}_1 : X_t \sim \begin{cases} \mathrm{F}_0, t = 1, 0, ..., \tau - 1 \\ \mathrm{F}_1, t = \tau, \tau + 1, ..., n \end{cases} \end{aligned} \tag{2.1}$$

La hipótesis nula H_0 indica que la variable X sigue una distribución F_0 durante toda la línea del tiempo, mientras que la hipótesis H_1 establece que la variable X sigue la distribución F_0 hasta el instante $t = \tau$, momento en que su distribución cambia a F_1. Las técnicas de CPA analizan la distribución de la variable X para aplicar un test de hipótesis que les permita decidir entre las hipótesis anteriores y determinar si existen *change points* y, de ser así, los instantes del tiempo más probables donde estos se han producido. [1].

El problema del CPA puede abordarse de muchas maneras y existen diversas técnicas con diferentes características que hacen que algunas de ellas sean más adecuadas dependiendo del tipo de datos que se maneje, la información que se tenga sobre ellos y el tipo de análisis que se quiera realizar. En las siguientes subsecciones se presentarán brevemente el abanico de técnicas de CPA disponibles actualmente, así como las ventajas e inconvenientes que presentan.

2.4.2. Técnicas de CPA paramétricas

Las técnicas de CPA paramétricas asumen que las variables del proceso estocástico a analizar siguen un tipo de distribución conocida y, a partir de esta asunción, contrastan las hipótesis del apartado anterior utilizando la función de verosimilitud o midiendo la desviación de los datos observados para detectar cambios de régimen.

Métodos basados en Gauss

La primera técnica que vamos a presentar se trata de la empleada en [5] para determinar si las neuronas PNs o las KCs del sistema olfativo de la langosta respondían a la estimulación o no. Esta técnica considera que se ha producido una respuesta cuando la tasa de disparo media de una neurona aumenta respecto a la que tiene en reposo (sin estimulación). Para ello, se asume que la tasa de disparo media durante en reposo sigue una distribución Gaussiana y los diferentes valores de tasa de disparo obtenidos se utilizan para calcular la desviación típica. Durante la fase de estimulación, si la tasa de disparos media supera en n veces la desviación típica, se considera que ha habido una respuesta.

Este método presenta el problema de determinar qué valor de n es el más adecuado para obtener resultados significativos, y de todas formas solo presenta una medida de cuánto se desvían las observaciones respecto al valor medio e ignora la información obtenida durante el periodo de estimulación, en lugar de emplearla para realizar un contraste entre ambas hipótesis. En el caso de no detectar respuesta, este método tampoco proporciona pruebas que esta no se haya producido en realidad. [7]

Métodos basados en la función de verosimilitud

Una primera aproximación para poder determinar si se ha producido un *change point* en una serie temporal aplicando la función de verosimilitud sería aplicando el test de Neyman-Pearson [8] calculando el ratio entre el valor de la verosimilitud de dos hipótesis dados la serie de datos X observados. La hipótesis $H_0 = \neg R$ se correspondería con la ausencia de respuesta durante X, y la serie $H_1 = R$ se correspondería con la existencia de una respuesta. De esta forma, el ratio se calcularía como

$$\lambda = \frac{L(H = H_1|X)}{L(H = H_0|X)}.$$

Sin embargo, para poder aplicar este test es necesario conocer las probabilidades a priori de cada hipótesis dados los datos $P(H_0|X)$ y $P(H_1|X)$. En el caso de muchos conjuntos de datos, la probabilidad $P(H_1|X)$ no puede obtenerse porque no se tiene certeza sobre si se ha producido una respuesta en el sistema o no.

Otros métodos como el propuesto en [9] intentan estimar la localización de los k *change points* de una serie temporal mediante la minimización de una función de coste basada en el cálculo de una función de contraste $J(\tau, X)$ que expresa cómo de bien se ajusta el modelo con τ *change points* a la serie observada. En este caso, la función de contraste se define como la función de verosimilitud logarítmica de una distribución normal, aunque los datos observados no se ajusten a dicha distribución.

2.4.3. Técnicas de CPA no paramétricas

Las técnicas no paramétricas no asumen que las variables del proceso sigan una distribución conocida y se basan en estimar la función de densidad de probabilidad de dichas variables

por medio de métodos no paramétricos como estimaciones obtenidas directamente de los datos o aplicando otras técnicas no paramétricas para obtener información sobre la distribución de probabilidad de los eventos, como *Bootstrapping* [10]. Este es el caso del método bayesiano propuesto, en el que no es necesario saber cómo se distribuyen los datos, sino que esta información se obtiene directamente de los datos de los experimentos y después se utiliza la ley de Bayes para comparar las distribuciones de probabilidad de los eventos en los dos regímenes. Si ambas distribuciones son similares, se considera que no ha habido un cambio de régimen y, si son diferentes por encima de cierto límite de probabilidad, se considera que se ha producido una respuesta en el sistema.

Por otro lado, existen técnicas muy potentes como la presentada en [11] que permite hallar todos los *change points* de una serie temporal a un coste computacional relativamente bajo. Esta técnica de eDevisive se basa en calcular una medida de divergencia entre los dos regímenes existentes en el sistema antes y después del *change point* considerado. Para poder hallar los diferentes *change point* de una serie temporal, este algoritmo utiliza una estrategia en la que la serie temporal se va dividiendo en dos intervalos en los que se aplica la medida de divergencia para determinar si el punto de división entre ambos es un *change point* o no.

3

Diseño y desarrollo

En el siguiente capítulo se detalla el método de CPA propuesto por el GNB, los requisitos y el diseño del programa desarrollado para implementarlo y algunas utilidades adicionales que se han diseñado para poder validarlo y compararlo con otros métodos.

3.1. Método bayesiano de CPA

El método de CPA propuesto por el GNB se presenta y explica detalladamente en [2]. Este método no paramétrico permite obtener una cota inferior de probabilidad de respuesta de un sistema ante un estímulo externo mediante la ley de Bayes, mejorando los resultados que pueden obtenerse mediante algunas técnicas de CPA paramétricas. Para poder aplicarse, es necesario definir y detectar en el sistema con el que se está trabajando algún cambio que pueda ser considerado un evento. En el caso de una neurona, los eventos se corresponderían con los disparos neuronales. El protocolo seguido para la realización de los experimentos y la obtención de los datos consiste en registrar los instantes del tiempo en los que se producen disparos neuronales para la duración total del experimento $T_{experimento}$ y estimular el sistema durante una ventana $\Delta t_{inicioestimulacion} = T_{estimulacin}$, donde $t_{inicioestimulacin}$ es el instante del tiempo donde se comienza a estimular el sistema y $T_{estimulacin}$ es la duración del periodo de estimulación, con el fin de determinar si se producen cambios en la distribución de los disparos con respecto al resto del experimento.

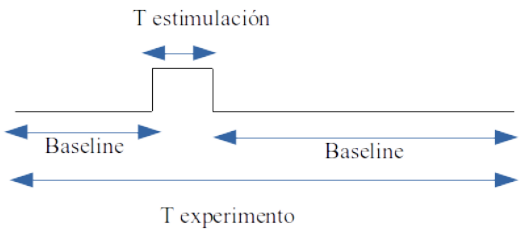


Figura 3.1: Protocolo seguido en la realización de los experimentos

De esta manera, para n pruebas realizadas en un experimento sobre el sistema en el que este

es estimulado, se puede obtener una serie de eventos $s_1, s_2, ..., s_n$, donde s_i se corresponde con el número de eventos registrados para la prueba n en una ventana de tiempo $\Delta t_p = P$ durante el experimento, donde el subíndice p indica el momento del tiempo de inicio de la ventana y P su duración.

A partir de la serie de eventos registrados y aplicando la ley de Bayes, puede calcularse la probabilidad de que no se haya producido una respuesta ($\neg R$) a un estímulo determinado como

$$P(\neg R|s_1, s_2, ..., s_n, estimulo) = \frac{P(s_1, ..., s_n|\neg R)P(\neg R|estimulo)}{P(s_1, ..., s_n|estimulo)}$$

En la expresión anterior, $P(s_1, ..., s_n|\neg R)$ puede obtenerse a partir de la actividad del sistema cuando está en reposo (*baseline*), es decir, cuando el estímulo específico no está presente, mientras que $P(s_1, ..., s_n|estimulo)$ puede obtenerse a partir de la actividad durante el periodo de estimulación o la ventana Δt_p que se esté utilizando. Por último, no podemos saber el valor de $P(\neg R|estimulo)$ ya que las probabilidades a priori son muy difíciles de medir en los sistemas nerviosos, debido a que no podemos saber con seguridad si el sistema está respondiendo o no ante un estímulo, pero se sabe que $P(\neg R|estimulo) \leq 1$. Teniendo lo anterior en cuenta, se obtiene:

$$P(\neg R|s_1, s_2, ..., s_n, estimulo) \leq \frac{P(s_1, ..., s_n|baseline)}{P(s_1, ..., s_n|estimulo)}$$

. De esta forma, la cota inferior de probabilidad de que el sistema responda ante un estímulo puede calcularse como:

$$P(R|s_1, s_2, ..., s_n, estimulo) \geq 1 - \frac{P(s_1, ..., s_n|baseline)}{P(s_1, ..., s_n|estimulo)}$$

. Para determinar si el sistema responde o no ante el estímulo, puede fijarse un umbral de probabilidad p_r, de tal manera que si se cumple $P(R|s_1, s_2, ..., s_n, estimulo) \geq p_r$, se considera que el sistema responde.

Finalmente, el estimador de la cota inferior de $P(R|s_1, s_2, ..., s_n, estimulo)$ puede expresarse como

$$\phi(R|s_1, ..., s_n) = 1 - \frac{P(s_1, ..., s_n|baseline)}{P(s_1, ..., s_n|estimulo)} \geq p_r$$

A la hora de obtener las probabilidades $P(s_1, ..., s_n|baseline)$ y $P(s_1, ..., s_n|estimulo)$, los sucesos $s_1, ..., s_n$ pueden considerarse o no independientes entre sí dependiendo del tipo de sistema que se esté analizando. En el caso de una neurona, unos sucesos pueden tener influencia sobre otros debido al aprendizaje, la reverberación de información, etc. A la hora de calcular las distribuciones de probabilidad, se pueden tener en cuenta diferentes relaciones de dependencia entre los sucesos. En el caso de sucesos independientes entre sí, como es el caso de los datos reales que se analizarán posteriormente [2], el estimador se puede calcular como:

$$\phi(R|s_1, ..., s_n) = 1 - \frac{\prod_{i=1}^{n} P(s_i|baseline)}{\prod_{i=1}^{n} P(s_i|estmulo)} \geq p_r$$

.

3.2. Modelo simple de generación de eventos

Con el propósito de poder realizar diferentes pruebas y validar las funcionalidades implementadas en el programa, se desarrolló en primer lugar un script en Python que permitiese generar

series de eventos similares a las obtenidas en los experimentos reales donde se estimulan neuronas durante cierto intervalo de tiempo. Para ello, se definen clases para codificar los estímulos que se aplicarán sobre el sistema y para generar los eventos en función de dichos estímulos.

Cada estímulo Θ_i tiene asociada una tupla de parámetros (α_i, k_i), donde α_i representa la fuerza del estímulo y k_i su desviación. Estos parámetros se distribuyen ambos según una distribución normal $N(\mu, \sigma)$. La desviación del estímulo es su capacidad para modificar la tasa λ a la que se generan los eventos en el sistema, mientras la fuerza modula el efecto de dicha desviación de forma que se introduzca cierto nivel de ruido en la serie de eventos. El sistema genera eventos siguiendo un proceso de Poisson con una tasa λ, ya que se ha comprobado que la distribución de disparos neuronales en el tiempo puede aproximarse bien por Poisson [12].

Para generar los eventos, se sigue el método propuesto en [13], calculando mediante la distribución exponencial asociada de tasa λ un número aleatorio que representa el tiempo hasta que se produce el siguiente evento en la serie, de manera que se obtienen una serie de los instantes del tiempo $t_1, ..., t_n$ con $t_i \leq T_{experimento}$ en los que se ha producido un evento. La tasa λ del proceso permanece constante mientras no hay estimulación. Durante el periodo de estimulación, el proceso se contamina según los parámetros α_i y k_i del estímulo Θ_i que se esté aplicando. De esta manera, se realiza una suma entre dos procesos de Poisson, $N_1(t)$ que se corresponde con el original de tasa λ_{N_1}, y un segundo proceso $N_2(t)$ con $\lambda_{N_2} = k_i * \lambda_{N_1}$. La suma de estos procesos se realiza dando a $N_1(t)$ más peso o menos peso que a $N_2(t)$ dependiendo del parámetro α_i del estímulo. Así, la suma puede expresarse como:

$$N_1(t)' = (1 - \alpha)N_1(t) + \alpha N_2(t)$$

. Para calcular $\lambda_{N_1'}$, el parámetro que nos interesa para poder seguir generando eventos en el nuevo régimen, teniendo en cuenta la expresión anterior, puede expresarse como:

$$\lambda_{N_1'} = (1 - \alpha)\lambda_{N_1} + \alpha\lambda_{N_2} = (1 - \alpha)\lambda_{N_1} + \alpha k_i \lambda_{N_1}$$

[14]

Como ejemplo de los datos generados por este modelo, se muestran varias gráficas de los datos obtenidos para diferentes valores de los parámetros α y k. En la Fig. 3.1 se muestran los disparos

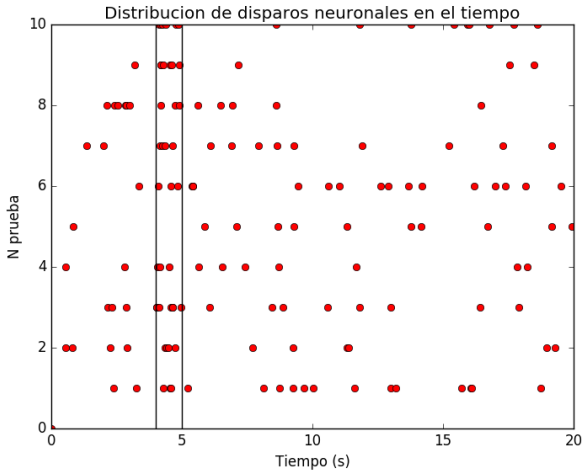


Figura 3.2: Distribución de disparos neuronales en un experimento de 20s de duración con 10 pruebas (eje Y). La neurona es estimulada entre los segundos 4 y 5. El estímulo utilizado tiene unos parámetros $\alpha = 0{,}65$ y $k = 8$.

neuronales cuando el sistema presenta una respuesta excitatoria ante el estímulo. Como puede

verse, la tasa de disparos entre los segundos $t = 4s$ y $t = 5$ es visiblemente mayor que para el resto de intervalos de tiempo. Si los datos anteriores se representan en un histograma de tiempo o PSTH (Fig. 3.2) donde se representa el número total de disparos en cada intervalo de tiempo de tamaño Δ, puede observarse de una forma más clara el aumento en la tasa de disparos para el intervalo de estimulación. La Fig. 3.3 es un ejemplo de respuesta inhibitoria ante un estímulo.

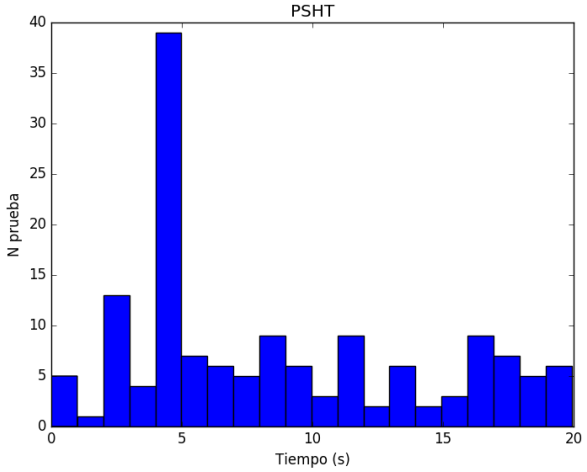


Figura 3.3: PSTH de un experimento de 20s de duración con 10 pruebas. La neurona es estimulada entre los segundos 4 y 5. El estímulo utilizado tiene unos parámetros $\alpha = 0{,}65$ y $k = 8$.

En este caso, la distribución de la fuerza de los estímulos sigue una distribución normal con $\sigma < 0$ para que en la mayoría de los casos $\lambda_{N_1'} < \lambda_{N_1}$ y se produzca un decremento en el número de disparos durante el periodo de estimulación:

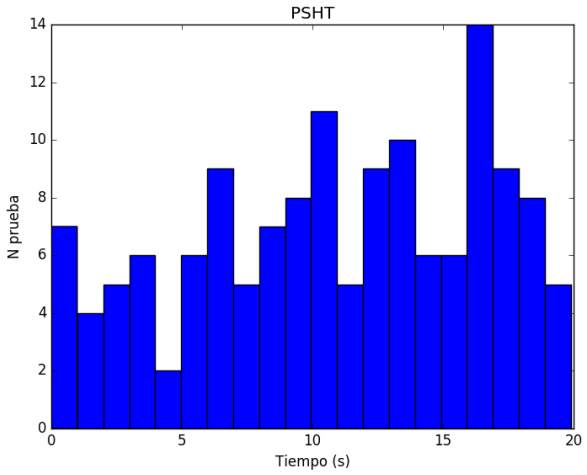


Figura 3.4: PSTH de un experimento de 20s de duración con 10 pruebas. La neurona es estimulada entre los segundos 4 y 5. El estímulo utilizado tiene unos parámetros $\alpha = -0{,}80$ y $k = 2$.

Por último, en la Fig. 3.4 se muestra el caso intermedio entre los dos anteriores en los que la tasa de disparos durante el periodo de estimulación y los periodos de reposo es muy similar. Será en este tipo de distribución donde sea más difícil determinar si se está produciendo o no una respuesta ante el estímulo y permitirá evaluar mejor la eficacia del método de CPA implementado.

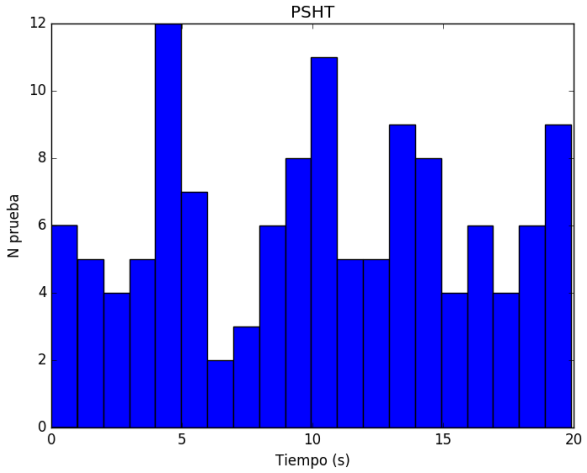


Figura 3.5: PSTH de un experimento de 20s de duración con 10 pruebas. La neurona es estimulada entre los segundos 4 y 5. El estímulo utilizado tiene unos parámetros $\alpha = -0{,}80$ y $k = 2$.

3.3. Requisitos del programa

Antes de desarrollar el programa, se especifican los requisitos que este debe cumplir, los cuales se enumeran a continuación:

- **RF1. Carga de datos:** el programa debe proporcionar las herramientas necesarias para poder importar datos de experimentos desde ficheros que cumplan un formato específico. Además de cargar los datos relacionados con los tiempos en los que se producen eventos durante los experimentos, debe permitir cargar otra información adicional como etiquetas asociadas a diferentes pruebas.
- **RF2. Selección y filtrado de experimentos:** el programador podrá cargar diferentes conjuntos de experimentos y, dentro de los mismos, seleccionarlos filtrando por etiquetas.
- **RF3. Análisis individual de experimentos:** el programa debe permitir hacer un análisis de cada experimento de forma individual. La información que debe proporcionar dicho análisis es la siguiente:
  - Tamaño de intervalo más adecuado para que la tasa de disparo se aproxime lo máximo posible a la tasa de disparo real subyacente en el experimento de forma que las probabilidades de respuesta obtenidas a partir del análisis de estos intervalos sean más ajustadas a la realidad.
  - Obtención de la probabilidad de respuesta por el método bayesiano para una ventana temporal dentro del experimento y un intervalo de baseline.
  - Evolución de la probabilidad de respuesta en función del tamaño de la ventana y en función del punto de inicio de dicha ventana.
  - Estimación del mejor límite inferior de probabilidad p_r para considerar que el sistema ha tenido una respuesta positiva ante el estímulo de tal manera que se reduzcan los falsos positivos por debajo de un porcentaje especificado.
  - Determinar si el sistema responde o no ante el estímulo para el experimento especificado mediante otro método diferente al bayesiano de tal forma que pueda establecerse una comparativa entre ambos. Este método será el basado en Gauss empleado en [5].

- **RF4. Análisis global del conjunto de experimentos:**las herramientas proporcionadas deben permitir al usuario analizar de forma global el conjunto de experimentos que seleccione. La información que debe proporcionarse es la siguiente:
  - Análisis de la sensitividad del sistema para un tamaño de ventana e intervalos de baseline determinados, de tal forma que puedan estudiarse aspectos relacionados con la codificación temporal.
  - Estimación del mejor límite inferior de probabilidad p_r global para reducir los falsos positivos por debajo de un porcentaje especificado.
- **RF5. Representación gráfica de datos y resultados:**el programa incluirá las herramientas necesarias para realizar las representaciones gráficas siguientes:
  - Representación de los eventos en el tiempo para diferentes pruebas (raster).
  - Representación de la distribución de los eventos en psths que muestren tanto el número total de eventos en un intervalo como la tasa de disparo media.
  - Representación de las probabilidades de respuesta en función del tamaño de ventana y del punto de inicio de la ventana, tanto en dos dimensiones como por medio de superficies.
  - Representación del gráfico de sensitividad del sistema suavizado.
  - Error cometido respecto a la distribución real de la tasa de disparo media en función del tamaño de intervalo seleccionado.
  - Diagrama de activación que muestre las neuronas que presentan o no presentan respuesta ante los diferentes estímulos.
- **RNF1: Modularidad:** la arquitectura del programa debe ser lo más modular posible de forma que sea fácil añadir nuevos componentes con el fin de que los usuarios futuros de la herramienta puedan implementar sus propios métodos de CPA, incluirlos y utilizarlos integrados con el resto de herramientas de análisis proporcionadas.
- **RNF2: Eficiencia:** debido a que es necesario procesar gran cantidad de datos y realizar muchas operaciones, el programa se implementará en Python al ser un lenguaje que incluye muchas utilidades y librerías eficientes para tratamiento de datos científicos, de las cuales se usarán numpy y matplotlib.

3.4. Diseño del programa

3.4.1. Diagrama de clases

La Fig. 3.5 muestra el diagrama de clases del programa de las herramientas desarrolladas y se ofrece una breve explicación de la funcionalidad implementada por cada clase.

- **DataLoader:** en esta clase se encuentran los métodos necesarios para procesar los ficheros con los datos de los experimentos y almacenarlos en estructuras que puedan ser manejadas por el resto de herramientas.
- **Experiment:** contiene los datos sobre los eventos de cada uno de los experimentos para las diferentes pruebas realizadas en forma de matrices.
- **ExperimentSet:** contiene varios experimentos agrupados con unas etiquetas y características comunes.

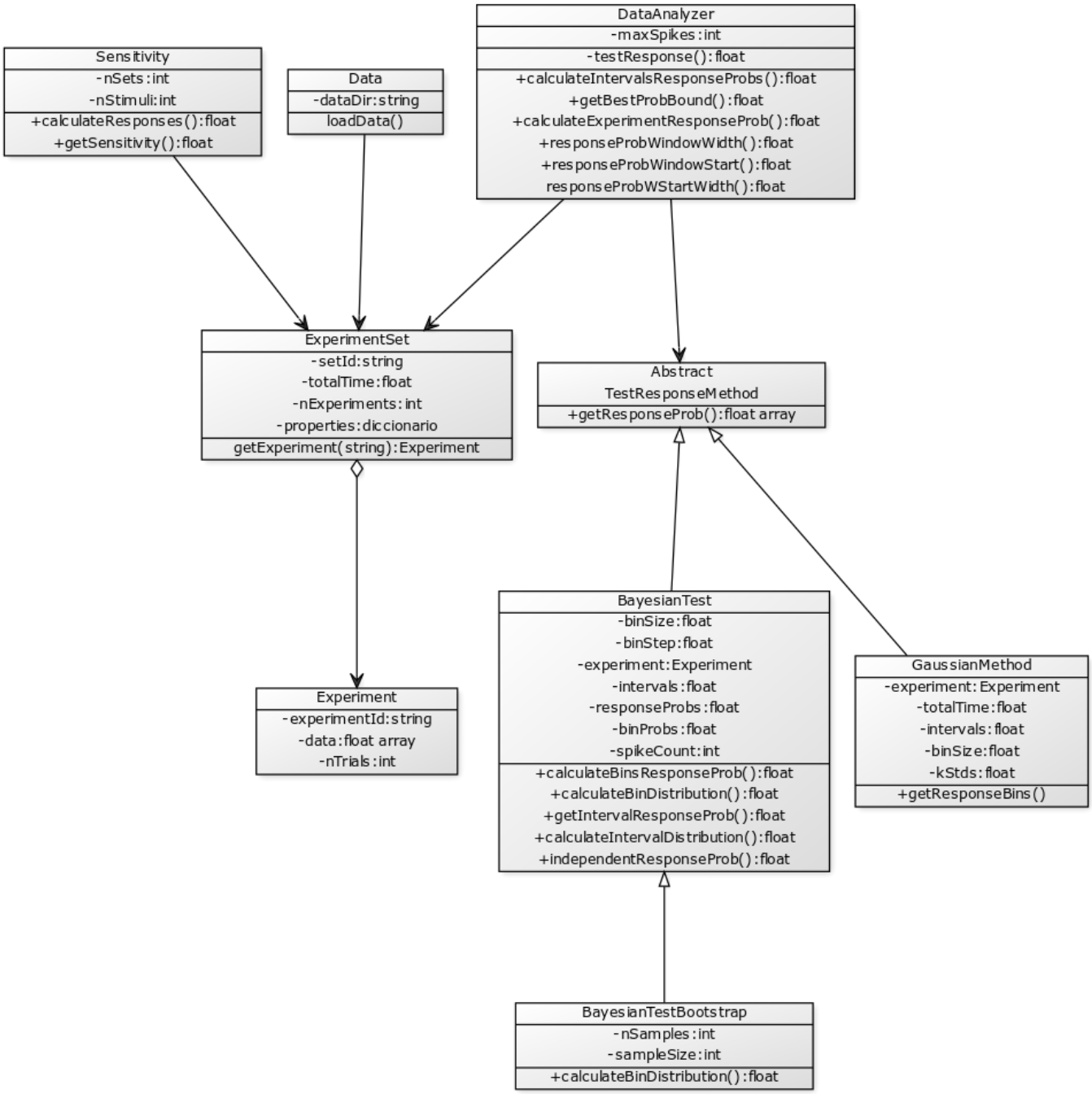


Figura 3.6: Diagrama de clases del programa de herramientas implementado

- **DataAnalyzer:** esta clase contiene los métodos necesarios para realizar el análisis individual de cada uno de los experimentos en un ExperimentSet, así como para calcular probabilidades de respuesta y proporcionar a otras herramientas la información que necesitan.
- **Sensitivity:** contiene los métodos necesarios para obtener el análisis global de todos los experimentos sobre el sistema y su sensitividad.
- **TestResponseMethod:** interfaz que deben implementar todos los métodos de CPA que se añadan al programa, de forma que puedan utilizarse de forma opaca al resto de las herramientas.
- **BayesianTest:** implementa el método de estimación de la probabilidad de respuesta bayesiano descrito en los apartados anteriores.
- **BayesianTestBootstrap:** variación del método anterior en el que se utiliza bootstrapping para mejorar la estimación de las distribuciones de probabilidad de los eventos.
- **GaussianMethod:** implementa el método de CPA basado en Gauss mencionado anteriormente para poder establecer una comparativa con la alternativa del bayesiano.

El código y más información acerca del sistema puede encontrarse en el Anexo B.

3.4.2. Formato y representación de los datos

El programa, en concreto por la clase DataLoader encargada de procesar los ficheros y cargarlos en memoria, acepta los datos de los experimentos en dos formatos utilizados comúnmente a la hora de almacenar los resultados de experimentos sobre sistemas en los que se tiene una serie temporal de eventos. Los experimentos suelen estar formados por una serie de repeticiones del mismo o pruebas, midiéndose los instantes del tiempo desde el inicio del experimento en los que se registra un evento.

De esta forma, el programa acepta como entrada ficheros que contienen los tiempos en los que se produjeron los eventos medidos desde el inicio del experimento o medidos desde el inicio de una prueba concreta. En ambos casos, son almacenados por la herramienta en listas de objetos Experimento que incluyen los eventos como matrices de dos dimensiones en las que la primera columna se corresponde con el número de prueba dentro del experimento y el la segunda, con el instante de tiempo relativo al inicio de la prueba. Además de esto, incluye otra información adicional como el identificador del experimento y el número de pruebas que contiene.

Cada grupo de experimentos correspondientes a un mismo sistema, como por ejemplo, a una misma neurona, se agrupa dentro de un ExperimentSet que consta de un identificador, el tiempo total de duración del experimento y un diccionario que asocia etiquetas con experimentos del conjunto, de forma que puedan seleccionarse y filtrarse. Las etiquetas asociadas a cada conjunto de experimentos pueden especificarse en el mismo fichero donde se incluyen los tiempos de los eventos o pueden incluirse en un fichero adicional de acuerdo a un formato establecido. Más detalles sobre el formato de los ficheros pueden consultarse en el Anexo B.

3.4.3. Implementación del análisis de datos

En esta sección se explica con más detalle algunos de los aspectos más relevantes sobre la implementación de varias de las funcionalidades más importantes del programa para permitir analizar los experimentos y estimar si se produce o no una respuesta del sistema.

Estimación del tamaño de intervalo óptimo para la estimación de las distribuciones de probabilidad

Existen muy diversas técnicas que pueden emplearse para estimar la densidad de probabilidad que siguen los disparos neuronales en el tiempo, tales como la estimación a través de histogramas, métodos kernel, verosimilitud o bootstrapping [15]. En este trabajo las estimaciones se realizarán directamente a través de los histogramas asociados a los experimentos combinados con bootstrapping por ser esta una técnica sencilla que ofrece buenos resultados.

Para el cálculo de la distribución de probabilidad de los eventos de forma que se capte su variación a lo largo de la serie temporal, es necesario dividir el tiempo total del experimento, T en intervalos de un tamaño fijo Δ, como se explicará en más detalle en el siguiente apartado.

La selección del valor de Δ no es trivial, ya que dependiendo del valor elegido, la distribución de probabilidad puede variar mucho. Por ejemplo, si se toma un Δ muy grande, se pierde la relación entre la tasa de disparo y el tiempo, mientras que si Δ es muy pequeño, la tasa de disparo presenta una gran variación, dificultando que se pueda captar la distribución de eventos subyacente. Lo ideal sería utilizar un valor de Δ que hiciese que la distribución resultante se aproximase lo máximo posible a la distribución real de la tasa de disparo [16].

Para resolver el problema de elegir el Δ más adecuado, podría recurrirse a los métodos más conocidos como fijar un número de intervalos $k = \sqrt{n}$, donde n es el número de eventos en el experimento, u otros como la fórmula de Sturges, Donaes, etc. que se basan en hacer suposiciones sobre la forma de la distribución de los datos. Sin embargo, el método de CPA basado en Bayes que el programa implementa no hace ninguna asunción sobre la distribución que siguen los datos, por lo que estos métodos para elegir el valor de δ no serían adecuados [17].

Por ello, para hallar el mejor Δ se ha elegido implementar el método presentado en [16], que se basa en minimizar la diferencia entre la tasa de disparo obtenida con un determinado valor de Δ, que se denominará $r_\Delta(t)$ y la tasa de disparo real durante el experimento, $r(t)$. Para calcular la diferencia entre $r_\Delta(t)$ y $r(t)$, se emplea el error cuadrático medio integrado, que se expresa como:

$$MISE(\Delta) = \frac{1}{T}\int_0^T E(r_\Delta(t) - r(t))^2 dt$$

El problema para poder calcular este error reside en que $r(t)$ es desconocido, pero se puede inferir a partir de los datos del experimento de tal forma que es posible derivar una función de coste para minimizar el MISE respecto a Δ a partir de la expresión anterior, como se demuestra en [17] asumiendo únicamente que los eventos suceden según un proceso de Poisson con una tasa $r(t)$ y son independientes entre sí. La función de coste resultante sería la siguiente:

$$C(\Delta) = \frac{2\overline{k} - v}{(n\Delta)^2}$$

En esta función, $\overline{k}$ representa la media del número de eventos por intervalo ($\overline{k} = \frac{1}{N_b}\sum_{i=1}^{N_b} k_i$, donde N_b es el número de intervalos) y v representa la varianza en el número de eventos por intervalo ($v = \frac{1}{N_b}\sum_{i=1}^{N_b}(k_i - \overline{k})^2)$).

Para obtener el mejor Δ para un experimento, se computa la función de coste $C(\Delta)$ para cada uno de sus posibles valores y se elige el que minimiza la función. En el caso de que la función no presente ningún mínimo, es indicativo de que no hay datos suficientes en el experimento y se debería revisar el protocolo que se está siguiendo en su realización para introducir más pruebas. La función de coste para un mismo valor de Δ se promedia desplazando el punto de inicio de los intervalos para suavizar la función y evitar grandes fluctuaciones en los valores obtenidos.

En el capítulo siguiente se analiza con más profundidad los resultados obtenidos mediante la aplicación de este método par el cálculo de Δ.

Estimación de la probabilidad de respuesta con el método Bayesiano

El cálculo de la probabilidad de respuesta con el método basado en Bayes propuesto se implementa en las clases BayesianTest y BayesianTestBootstrap. A continuación se explican los cálculos que se llevan a cabo en ellas.

Para calcular la probabilidad de respuesta del sistema ante un determinado estímulo en una ventana temporal de estimación $\Delta_{t_{inicio}} = duracion$ y una ventana de baseline $\Delta_{t_{baseline}} = duracion$, en primer lugar es necesario como se explicó en apartados anteriores, obtener los valores de $P(s_1, ..., s_n|baseline)$ y $P(s_1, ..., s_n|ventana)$ a partir de la distribución de los eventos en el tiempo. Para calcular la distribución de los eventos, el tiempo T_{prueba} de las pruebas del experimento se divide en intervalos de tamaño Δ con una distancia d entre ellos. Para cada uno de estos intervalos, se cuantifica el número de eventos $s_1, ..., s_n$ que se han producido para cada prueba. Después, se calcula la probabilidad $P(s_i|intervalo) = \frac{s_i}{N_p}$, donde N_p es el número de pruebas en el experimento. Estas probabilidades se almacenan en una matriz para consultarse posteriormente de forma eficiente.

Con las distribuciones de probabilidad calculadas para cada intervalo, puede obtenerse su probabilidad de respuesta. Para ello, se compara la distribución de un intervalo concreto con la distribución de los intervalos que pertenecen al periodo de baseline del experimento, es decir, se compara la distribución $P(s_1, ..., s_n|intervalo)$ con $P(s_1, ..., s_n|baseline)$. La distribución $P(s_1, ..., s_n|baseline)$ se obtiene promediando las probabilidades $P(s_i|intervalo')_{baseline}$ de cada intervalo perteneciente al periodo de baseline. Una vez obtenidas las dos distribuciones, se calcula el límite inferior para la probabilidad de respuesta mediante la expresión $P(R|s_1, s_2, ..., s_n, intervalo) \geq 1 - \frac{P(s_1,...,s_n|baseline)}{P(s_1,...,s_n|intervalo)}$. El valor resultante se almacena asociado al intervalo al que corresponde.

Finalmente, para calcular la probabilidad de respuesta $P(R|s_1, s_2, ..., s_n, ventana)$ de la ventana de estimación $\Delta_{t_{inicio}} = duracion$ definida por el usuario, se obtienen los intervalos que están incluidos en la ventana y se promedia su probabilidad de respuesta.

Una de las grandes ventajas que ofrece el método expuesto es que el usuario tiene total libertad a la hora de introducir las ventanas de estimación y de baseline, permitiéndole explorar todas las combinaciones de intervalos posibles, algo muy útil para medir tiempos de retardos entre el inicio de la estimulación y la respuesta del sistema, las posibles reverberancias de información o la codificación temporal.

Además de la estimación de las distribuciones de probabilidad directamente a partir de los datos disponibles, se ha implementado otra versión del método anterior en la que se utiliza la técnica de Bootstrap [10] para mejorar la estimación de dichas distribuciones. Este método es no paramétrico, por lo que es aplicable al método bayesiano implementado, y permite obtener más información sobre la distribución a partir de una muestra pequeña como sucede en el caso de los experimentos que nos ocupan. La idea central del bootstrapping es generar n muestras de Bootstrap de tamaño m a partir de una muestra disponible, tomando valores con repetición de la misma.

Para aplicar bootstrapping al método anterior, una vez se han calculado el número de eventos por prueba $s_1, ..., s_n$ del experimento, se toman n muestras de Bootstrap de tamaño $N_{pruebas}$ a partir de dicha muestra. Para cada una de las muestras resultantes, se calcula su distribución de probabilidad $P(s_i|intervalo)$ y se calcula la distribución final realizando el promedio de todas las distribuciones anteriores. En el siguiente capítulo se analiza el impacto de la utilización de bootstraping en el cálculo de las probabilidades de respuesta.

Minimización de falsos positivos

Los falsos positivos en el contexto del método desarrollado se producen cuando se considera como una respuesta ante un estímulo específico una distribución de eventos que en realidad no lo es ya que, cuando se produjo, el sistema no estaba siendo estimulado. Esto es, un falso positivo se produce cuando el método considera como respuesta una distribución de eventos situada en el periodo de baseline.

Para minimizar el número de errores de este tipo que se puedan cometer, es necesario fijar un límite inferior p_r para el estimador $\phi(R|s_1, ..., s_n) \geq p_r$ lo suficientemente elevado como para que el número de falsos positivos no supere cierto margen definido por el usuario [18].

Para obtener el límite p_r, las clases DataAnalyzer y Sensitivity disponen de métodos que calculan dicho límite tanto para experimentos indiviudales como para el conjunto total de experimentos disponibles. Para ello, una vez se ha calculado la probabilidad de respuesta para cada intervalo de tamaño Δ, se toma la probabilidad de respuesta más pequeña como p_r y se comprueba cuántos intervalos pertenecientes a los intervalos de baseline definidos por el usuario presentarían una respuesta positiva. Si el número de falsos positivos es menor que el porcentaje especificado por el usuario, se elige esa probabilidad como p_r más adecuado. Si no, se prueba con el valor de probabilidad inmediatamente mayor, hasta encontrar uno que reduzca los falsos positivos por debajo de la tasa especificada.

Método basado en Gauss

Con el propósito de comparar el método Bayesiano propuesto con otro método de CPA paramétrico más sencillo, se ha implementado el utilizado en [5] basado en Gauss y en el test de Fisher. Este método se basa en calcular la tasa de disparo media durante los intervalos de baseline del experimento y compararla con la tasa de disparo media durante el periodo de estimulación del sistema. La condición para afirmar que se ha producido una respuesta es que la tasa de disparo media durante el periodo de estimulación supere en n veces la desviación estándar de la tasa de disparo durante el baseline. El valor de n elegido es arbitrario y se deja al criterio del usuario, lo que hace que este método sea poco riguroso. A pesar de ello, es uno de los más utilizados a la hora de analizar experimentos, y por ellos se ha elegido implementarlo.

Sensitividad del sistema

Además del análisis individual que puede realizarse sobre cada uno de los experimentos, también es interesante poder hacerse una idea de la respuesta global de todo el sistema ante los estímulos presentados. Para ello, en la clase Sensitivity se implementan métodos que permiten analizar la sensitividad del sistema de forma que pueda determinarse si este presenta una tendencia generalista o más específica en su respuesta hacia estímulos, así como aspectos de codificación temporal.

La sensitividad del sistema puede definirse como la probabilidad de que una neurona reaccione ante n estímulos de un total de N, es decir, $P(n|N)$ [2]. Por tanto, para calcular la sensitividad del sistema, se calcula para cada neurona y cada experimento realizado el número de estímulos ante los que esta presenta respuesta para cierto límite inferior p_r y se divide entre la cantidad total de estímulos. Los valores de $P(n|N)$ para los diferentes valores de n se almacenan para poder ser representados gráficamente y suavizados mediante una Gaussiana.

En el siguiente capítulo la sensitividad se utilizará para poder extraer importantes conclusiones sobre las características de los dos tipos de neuronas cuyos datos se van a analizar a modo de validación.

4

Experimentos y análisis de resultados

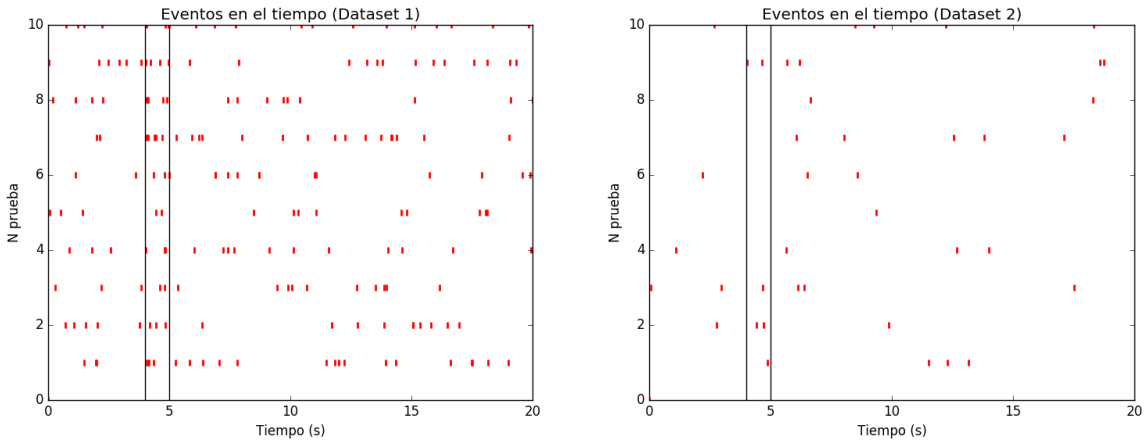
En el presente capítulo se presenta la validación de la metodología desarrollada mediante su aplicación a datos generados mediante un modelo matemático sencillo y los resultados obtenidos y el análisis de los mismos tras aplicarlo a datos reales procedentes del sistema olfativo de la langosta.

4.1. Validación de la funcionalidad mediante datos generados por el modelo

En esta sección se muestra la aplicación de las funcionalidades desarrolladas a conjuntos de datos obtenidos mediante el modelo de generación de datos que se explicó en el capítulo anterior, sección 3.2., con el fin de disponer de datos controlados que sirviesen como validación de las funcionalidades que se iban implementando. Los datos consisten en experimentos con una duración de $T_{prueba} = 20s$ y $N_{pruebas} = 10$. La ventana de estimulación es $\Delta_4 = 1s$ y los estímulos presentados pueden desencadenar tanto respuesta excitatorias como inhibitorias. Se mostrarán los resultados obtenidos al aplicar los algoritmos para la selección del tamaño de intervalo más adecuado, la estimación de la respuesta o ausencia de ella y la selección de un umbral p_r de tal forma que se reduzcan los falsos positivos por debajo de una tasa especificada.

4.1.1. Estimación del tamaño de intervalo óptimo para la estimación de la distribución de probabilidad

Con el fin de validar el funcionamiento del algoritmo desarrollado para elegir el tamaño de intervalo Δ de los histogramas más adecuado para calcular las distribuciones de probabilidad de los eventos en el tiempo, se generaron mediante el modelo desarrollado dos conjuntos diferentes de datos. El primero de ellos es un proceso de Poisson con una tasa $\lambda = 0{,}9$, por lo que se producirán muchos más eventos en el tiempo, y el segundo con una tasa $\lambda = 0{,}3$, con lo que el número de eventos en total será más reducido. Ambos conjuntos de datos se muestran en la Fig. 4.1. El objetivo es comprobar que el tamaño Δ elegido en el caso del primer conjunto de datos es más pequeño para poder capturar la variación de la distribución en el tiempo al haber mayor densidad de eventos, mientras que en el segundo conjunto, el Δ elegido debería ser más



(a) Dataset 1 con $\lambda = 0{,}9$ (b) Dataset 2 con $\lambda = 0{,}3$

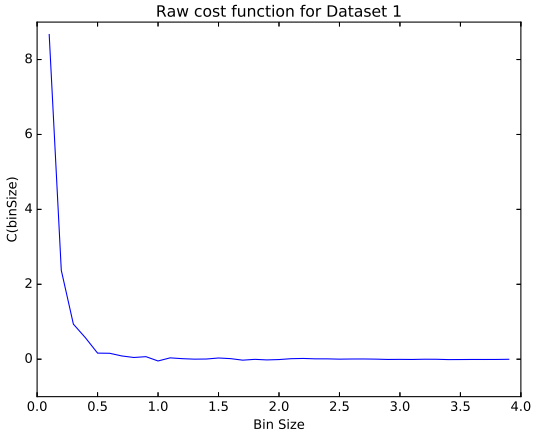
Figura 4.1: Datasets generados para la validación

pequeño debido a la menor densidad de eventos, según predice el método implementado (ver Sección 3.4.3). Tras calcular el valor de la función de coste utilizada por el método (ver Sección 3.4.3), $C(\Delta)$, para diferentes valores de Δ entre [1s-4s] con un incremento de 0.1s comenzando en $\Delta = 0{,}1$, en la Fig. 4.2 se muestra su representación gráfica para ambos conjuntos de datos en su versión sin suavizado y suavizada mediante el cálculo del promedio.

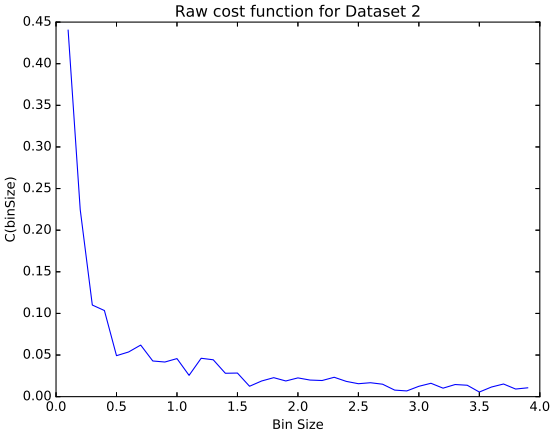
Como puede verse, la versión sin suavizado de $C(\Delta)$ (figuras a y b) presenta una gran fluctuación, sobre todo para los valores pequeños de Δ, lo que podría inducir a errores a la hora de calcular el mínimo de la función, por lo que realizar el suavizado es indispensable. Ambas funciones presentan un mínimo absoluto, lo que es indicativo de que los experimentos cuentan con datos suficientes.

En el caso del primer conjunto de datos, el valor recomendado de $\Delta_1 = 1{,}0$, aunque también servirían valores más pequeños dado que la función de coste decrece considerablemente y permanece más o menos constante a partir de $\Delta_1 = 0{,}5$ lo que está de acuerdo con la alta densidad de los eventos, ya que se necesitan intervalos pequeños para captar bien su variación. En el caso del segundo conjunto, el valor recomendado de $\Delta_2 = 3{,}5$, aunque también servirían valores más pequeños por la misma razón que en el caso anterior. Aún así este valor resulta adecuado dados los datos, ya que los intervalos deben ser más grandes dada la poca densidad de eventos que contiene. En la Fig. 4.3 pueden verse los PSTHs para ambos conjuntos con el tamaño de intervalo Δ seleccionado:

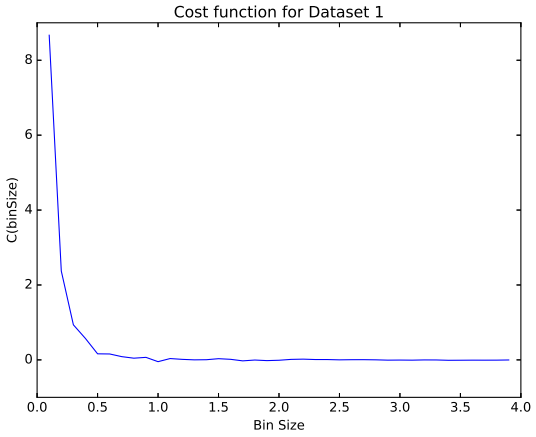
En el caso del conjunto de datos con $\lambda = 0{,}3$, donde la actividad del sistema es muy baja en todo momento, puede discutirse que el tamaño Δ óptimo elegido por el algoritmo es demasiado grande en comparación con el intervalo de estimulación que solo dura 1s y, por tanto, la respuesta que pudiese presentar el sistema durante este periodo se pierde, ya que queda camuflada con el resto de eventos dentro del intervalo de mayor tamaño. En el caso del conjunto de datos de $\lambda = 0{,}3$, la tasa de disparo apenas varía entre el intervalo de estimulación y el baseline, y eso puede explicar la elección de un Δ grande, ya que el método utilizado intenta aproximar la forma del histograma a la tasa de disparo del experimento, por lo que si la respuesta fuese más significativa durante el intervalo de estimulación, el valor de Δ se reduciría para captar esta variación en la tasa de disparo. Para comprobar si el método se comporta de esta manera, se genera un tercer conjunto de datos con $\lambda = 0{,}3$ y una respuesta mucho más clara que para el conjunto analizado previamente. Para este nuevo conjunto, el algoritmo devuelve $\Delta = 1{,}8$ como tamaño óptimo, un valor considerablemente menor. En la Fig. 4.4 puede observarse la distribución de eventos de este conjunto, su función de coste $C(\Delta)$ y el PSTH con Δ óptimo.



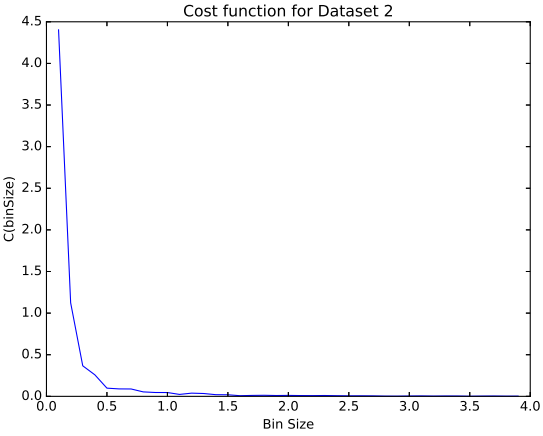
(a) $C(\Delta)$ sin suavizar para el dataset 1



(b) $C(\Delta)$ sin suavizar para el dataset 2

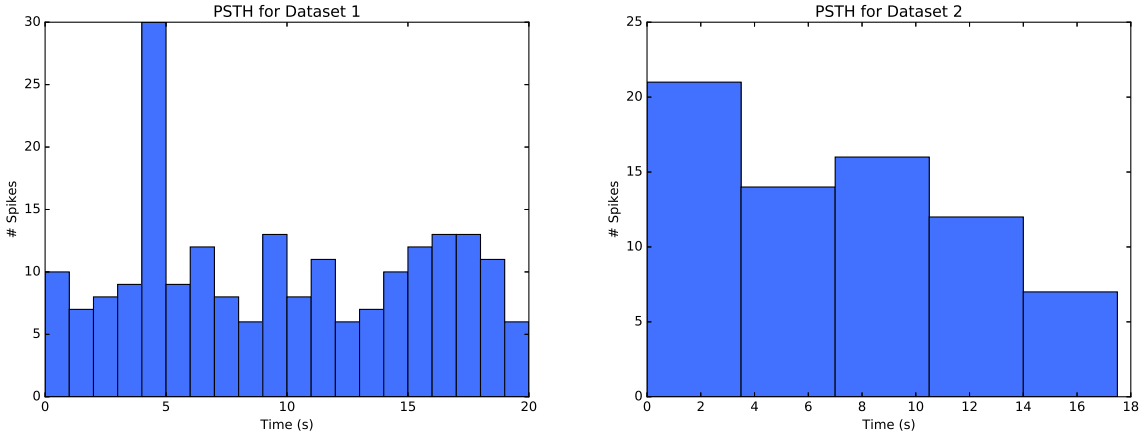


(c) $C(\Delta)$ suavizado para el dataset 1



(d) $C(\Delta)$ suavizado para el dataset 2

Figura 4.2: $C(\Delta)$ para el dataset 1 y el dataset 2



(a) PSTH del dataset 1 con $\Delta = 1{,}0$ (b) PSTH del dataset 2 con $\Delta = 3{,}5$

Figura 4.3: PSTHs de los datasets con los valores de Δ seleccionados

Como puede verse en el PSTH del experimento, aunque el Δ seleccionado sea mayor que la duración del tamaño de estimulación, sigue siendo perfectamente apreciable el aumento de la tasa de disparo en el intervalo que incluye al de estimulación. Sin embargo, en casos en los que la respuesta sea menos clara, es recomendable elegir tamaños Δ lo suficientemente pequeños para que esta no quede absorbida dentro del intervalo, aunque no se corresponda con el Δ óptimo dado por el método para adaptarse lo mejor posible a la tasa de disparo. Lo más adecuado sería escoger valores de Δ una vez la función de coste $C(\Delta)$ ha decrecido y se ha estabilizado.

4.1.2. Validación de la estimación del límite inferior de probabilidad de respuesta por medio del método Bayesiano

Con el fin de validar el cálculo de la probabilidad de respuesta con el método Bayesiano propuesto, se han generado con el modelo de generación de datos tres conjuntos con la misma tasa de disparo λ. La diferencia entre ellos es que el primer conjunto D_1 presenta una respuesta excitatoria ante el estímulo, el conjunto D_2 presenta una clara respuesta inhibitoria ante el estímulo y, finalmente, el conjunto D_3 presenta una respuesta excitatoria mucho menos pronunciada que D_1. Estos tres conjuntos nos permitirán comprobar si el método propuesto capta de manera correcta los diferentes tipos de respuesta que un sistema como una neurona puede presentar. En la Fig. 4.5 pueden verse representados los PSTHs de los tres conjuntos:

El siguiente paso es aplicar el método bayesiano a los datos. Para ello, se ha elegido un tamaño de intervalo $\Delta = 0{,}5s$, una ventana de estimulación $\Delta_4 = 1s$ que se corresponde con el periodo de estimulación de los sistemas y unos intervalos de baseline $[0s - 4s]$ y $[10s - 20s]$, que se corresponden con los primeros segundos del experimento antes de la estimulación y los últimos, descartando los instantes inmediatamente posteriores al periodo de estimulación para evitar que el baseline se contamine con reverberancias de información o actividad de respuesta ante el estímulo residual.

Para los tres conjuntos, la representación de la probabilidad de respuesta en función del tamaño de la ventana de estimación y del punto de inicio de la misma se muestra en la Fig. A.1. Las distribuciones de probabilidad se han estimado directamente a partir de los datos en la fila superior y en la inferior utilizando Bootstrap con un cantidad de muestras de $N_{muestras} = 100000$ y un tamaño de muestras de $N_{pruebas} = 10$.

En el caso de los conjuntos D_1 y D_2, el límite inferior de probabilidad de respuesta

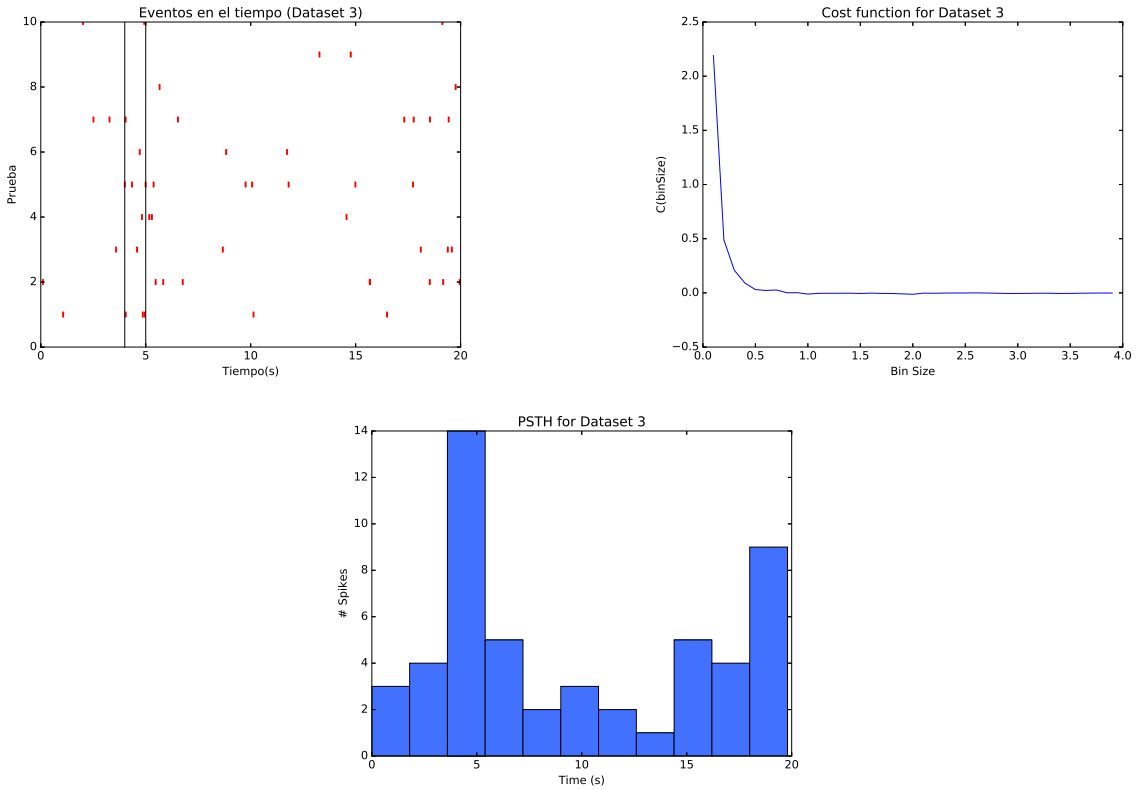


Figura 4.4: Distribución de eventos, función de coste y PSTH con $\Delta = 1{,}8$ para el dataset 3

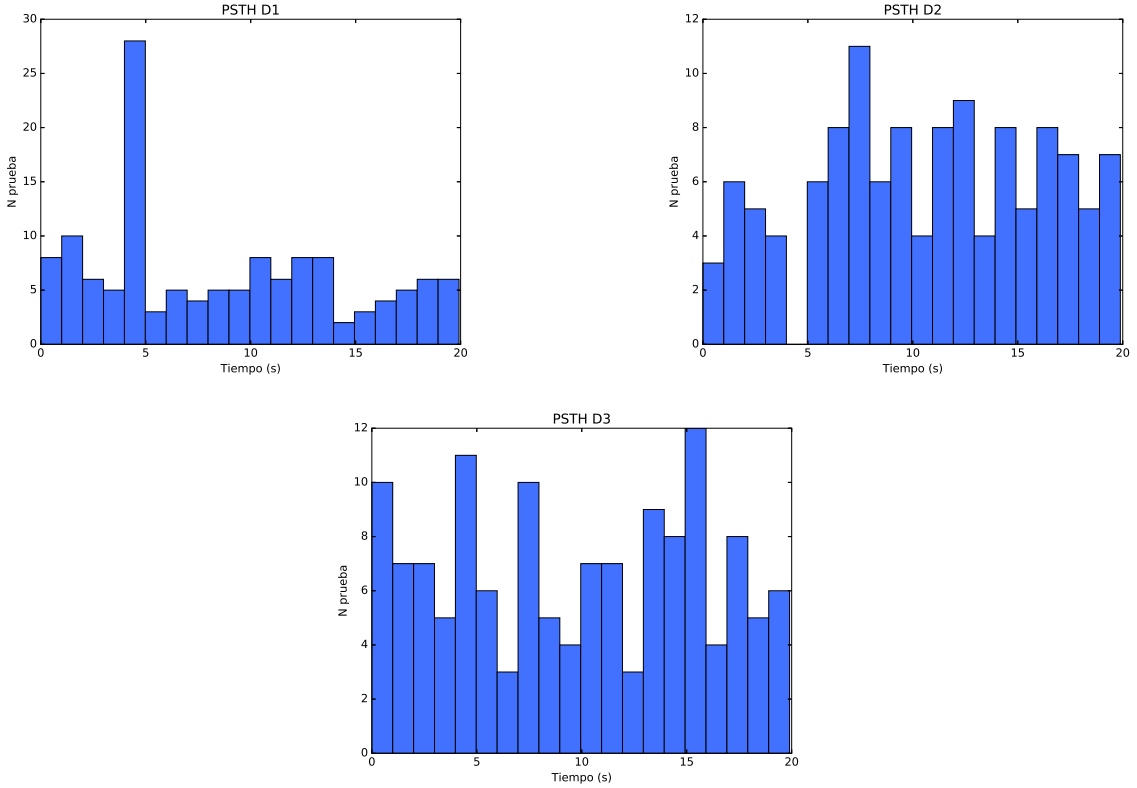


Figura 4.5: PSTHs de los datasets generados para la validación del método Bayesiano

$P(R|s_1, s_2, ..., s_n, estimulo)$ para la ventana que coincide con el periodo de estimulación del sistema $\Delta_4 = 1s$, la probabilidad de respuesta se eleva hasta alcanzar un valor de $P(R|\Delta) = 1{,}0$, lo que indica que el método es capaz tanto de captar respuestas excitatorias como respuestas inhibitorias ante los estímulos. Este comportamiento se justifica si observamos la expresión mediante la cual se calcula la probabilidad de respuesta $P(R|s_1, s_2, ..., s_n, estimulo) \geq 1 - \frac{P(s_1,...,s_n|baseline)}{P(s_1,...,s_n|estimulo)}$. En el caso de una respuesta excitatoria, la distribución $P(s_1, ..., s_n|estimulo)$ contendrá valores de probabilidad $P(s_i|estimulo)$ más altos para s_i más grandes, ya que durante el periodo de estimulación se incrementa la tasa de disparo, mientras que la distribución mientras que la distribución $P(s_1, ..., s_n|baseline)$ contendrá valores $P(s_i|estimulo)$ más altos para s_i pequeños, ya que la tasa de disparo es menor. En el caso de la respuesta inhibitoria, el mismo fenómeno sucede a la inversa. En ambos casos, al realizar la comparación de ambas distribuciones, $\frac{P(s_1,...,s_n|baseline)}{P(s_1,...,s_n|estimulo)} \sim 0$ y, por tanto el límite inferior de probabilidad de respuesta $1 - \frac{P(s_1,...,s_n|baseline)}{P(s_1,...,s_n|estimulo)} \sim 1{,}0$. Como puede observarse en las superficies representadas en la Fig. A.1, a medida que nos alejamos en el tiempo del intervalo de estimulación, el límite inferior de probabilidad de respuesta va cayendo a valores más bajos por debajo de 0,9.

En el caso del conjunto D_3, $P(R|s_1, s_2, ..., s_n, estimulo)$ no alcanza un máximo tan claro para la ventana coincidente con el intervalo de estimulación como en el caso de los datasets anteriores. Esto es así porque la tasa de disparo del periodo de estimulación y de baseline no difiere de forma lo suficientemente significativa y, por tanto, las dos distribuciones $P(s_i|ventana)$ y $P(s_1, ..., s_n|baseline)$ son muy parecidas, haciendo que $\frac{P(s_1,...,s_n|baseline)}{P(s_1,...,s_n|ventana)}$ alcance un valor más elevado que en los dos casos anteriores.

Por otro lado, vale la pena observar cómo $P(R|ventana)$ va descendiendo a medida que nos alejamos en el tiempo del periodo de estimulación, tal y como ocurriría en un sistema real al irse disipando el ascenso o descenso del nivel de actividad inicial ante el estímulo para volver al estado inicial.

Otra prueba interesante que puede realizarse para comprobar la corrección del método consiste en tomar como ventana los intervalos pertenecientes al baseline y compararlo con otro intervalo de baseline. En este caso, las dos distribuciones $P(s_i|ventana)$ y $P(s_1, ..., s_n|baseline)$ deberían muy similares y, por tanto, $\frac{P(s_1,...,s_n|baseline)}{P(s_1,...,s_n|estimulo)} \sim 1{,}0 \implies P(R|s_1, s_2, ..., s_n, ventana) \geq 0{,}0$.

Los resultados devueltos por el método para los conjuntos de datos, que pueden observarse en la Fig. A.2, cumplen con lo esperado también en este caso, ya que en para todos ellos, $P(R|s_1, s_2, ..., s_n, ventana)0{,}8$.

Por último, cabe destacar que, como puede observarse en las diferentes figuras de esta sección, la utilización de bootstrapping para el cálculo de las distribuciones de probabilidad en comparación con la estimación directa a partir de los datos disponibles no introduce una variación significativa en las mismas, al tratarse de muestras pequeñas con un rango pequeño de variación en los valores que pueden tomar los datos. Se ha calculado la diferencia promedio entre las distribuciones de probabilidad calculadas utilizando bootstrap y prescindiendo de él y para los tres conjuntos de datos, esta diferencia es del orden de 10^{-15}.

4.1.3. Validación de reducción de falsos positivos

Para validar el algoritmo de reducción de falsos positivos (ver Sección 3.4.3), tomamos el dataset D_3 y realizamos una representación gráfica con los valores de p_r considerados y el número de falsos positivos asociado a cada uno de ellos. Como puede verse en la Fig. 4.6, los valores obtenidos para p_r, que se encuentran representados mediante las líneas verticales, son $p_r = 0{,}9953$ para el 1 % de falsos positivos, $p_r = 0{,}9848$ para el 5 % y $p_r = 0{,}8942$ para el 10 % dejan a su izquierda el porcentaje seleccionado de falsos positivos.

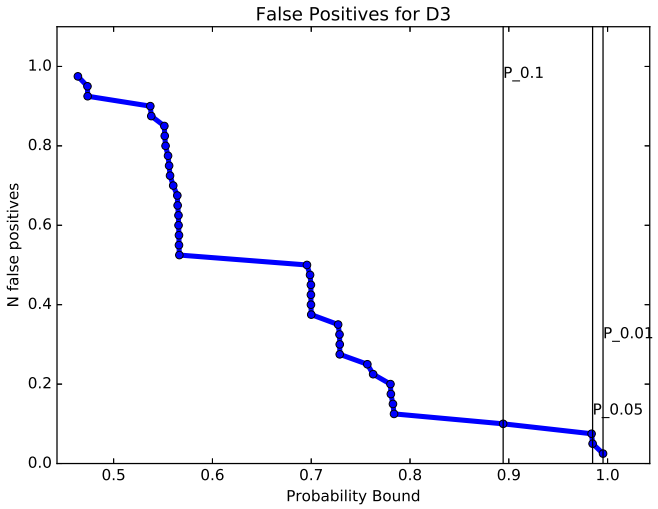


Figura 4.6: Porcentaje de falsos positivos en función de p_r

4.2. Aplicación del método a datos reales de células KCs y PNs

En esta sección se presentan los resultados obtenidos tras realizar un análisis con el programa desarrollado de los datos de diferentes experimentos sobre células KCs y PNs del sistema olfativo de la langosta. En primer lugar, se describirán las características de ambos conjuntos y después se pasará a la presentación de los resultados y su análisis comparativo.

4.2.1. Descripción de los conjuntos de datos

Los dos conjuntos de datos de los que se dispone incluyen diferentes experimentos sobre células KCs y PNs. Sus características más importantes se describen a continuación:

1. **Datos PNs:** estos datos se corresponden con varios experimentos realizados en [5] sobre 14 neuronas de proyección situadas en el lóbulo antenal de la langosta. Estos experimentos no se habían analizado nunca con el método bayesiano propuesto. Los insectos fueron expuestos a tres odorantes distintos a cinco concentraciones diferentes durante periodos de estimulación de $\Delta_{t=2s} = 1s$. La duración total de cada prueba del experimento es $T_{prueba} = 15s$, habiendo un total de $N_{pruebas} = 15$ pruebas por experimento. El intervalo recomendado para considerar como baseline, teniendo en cuenta lo anterior, serían $\Delta_{t=7s} = 8s$.

2. **Datos KCs:** los datos también se han obtenido de [5] y fueron analizados en [2]. Este conjunto de experimentos fue realizado sobre 43 células de Kenyon del cuerpo fungiforme de la langosta. Estas fueron expuestas a diecisiete olores diferentes a la misma concentración, alternando con algunos experimentos de control en los que no se les exponía a ningún odorante y otras en las que se les presentaba un estímulo luminoso, ya que este tipo de células parecen poder presentar respuestas si las condiciones de luminosidad varían. Los experimentos que se han seleccionado para analizar son en los que se realizó la presentación de odorantes. La duración total de cada prueba del experimento es de $T_{prueba} = 20s$, realizándose $N_{pruebas} = 10$ pruebas por experimento. Las neuronas fueron expuestas al estímulo en una ventana de estimulación $\Delta_{t=4s} = 1s$, por lo que el periodo recomendado de baseline sería $\Delta_{t=8s} = 12s$.

Cuadro 4.1: Características de los datos

	PNs Dataset	KCs Dataset
Nº de neuronas	43	14
Pruebas/Experimento	15	10
Tiempo(s)/Prueba	15	20
Intervalo estimulación	[2s-3s]	[3s-4s]
Intervalos baseline	[0s-2s][10s-15s]	[8s-20s]
Nº odorantes	3	17
Nº concentraciones	5	1
Nº total estímulos	15	17

En el cuadro 4.1 se muestran comparativamente la características de ambos conjuntos de datos a modo de resumen y para consulta rápida.

4.2.2. Aspectos generales

La figura A.3 muestra la representación de los eventos correspondientes a algunos experimentos que se han seleccionado como muestra de las características generales que presentan todos ellos, con sus PSTHs asociados.

El tamaño de los intervalos del histograma, Δ, se ha elegido con el algoritmo desarrollado (ver Sección 3.4.3) y la representación de la función de coste $C(\Delta)$ se puede ver en la Fig. A.4. Para los experimentos realizados sobre los KC's, se ha tomado un $\Delta = 1{,}0$ dado que es un valor cercano a lo que recomendaba el algoritmo, y para los PN's, $\Delta = 0{,}5$.

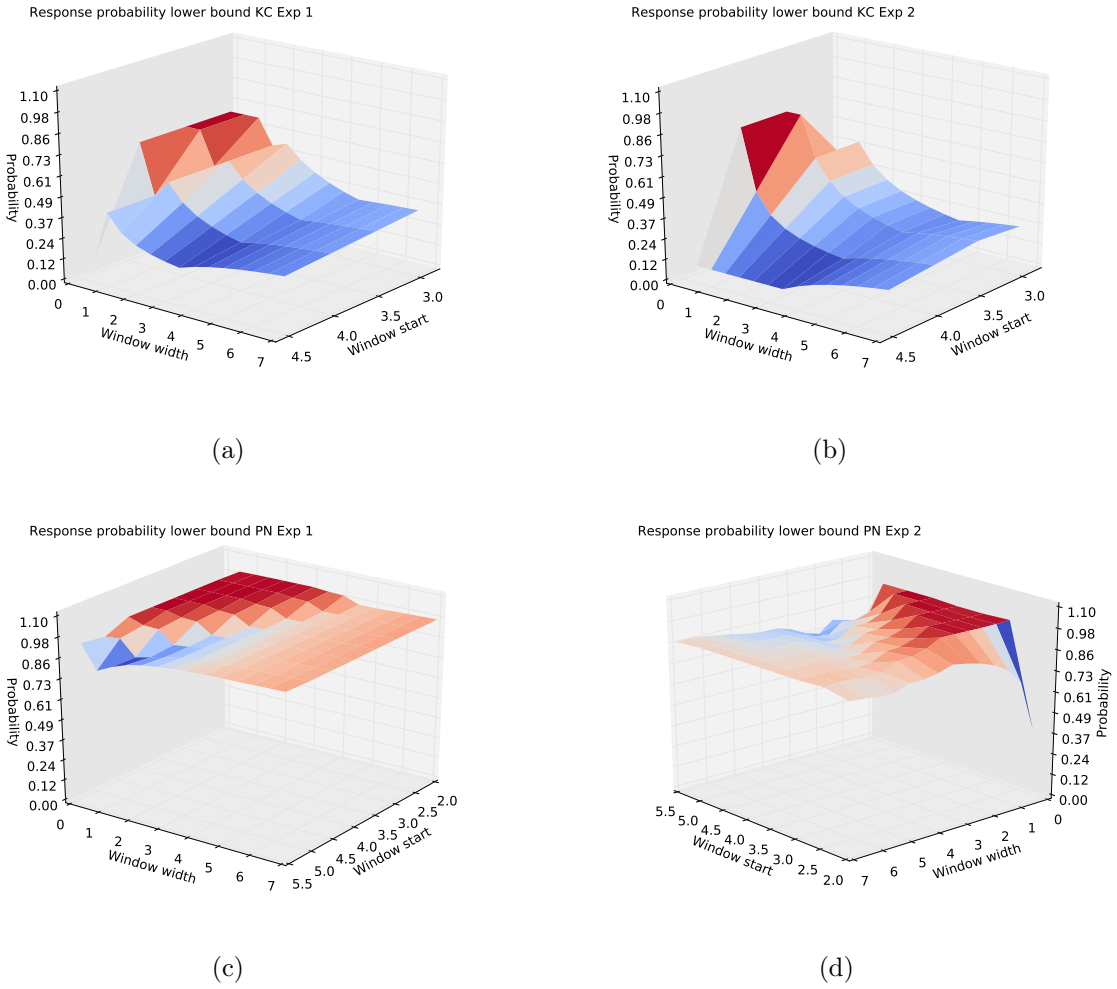
Como puede verse en la figura A.3, tasa de disparo de las KCs durante todo el tiempo del experimento es muy baja y apenas se registra actividad ni durante el periodo de baseline ni siquiera, en algunos casos, durante el período de estimulación, por lo que determinar si se produce una respuesta en estas neuronas constituye un problema difícil. Al contrario de lo que ocurre con las KCs, las PNs registran mucha más actividad constantemente, tal y como se indica en [5].

Otro aspecto relevante a destacar que puede entreverse en la representación de los datos anterior es que las neuronas PNs parecen presentar un código de respuesta complejo en el tiempo, ya que parecen intervenir mecanismo inhibitorios que cesan la actividad desde que se les presenta el estímulo hasta que se vuelven a registrar disparos neuronales de una manera intensa. Este aspecto se tratará más en detalle en la sección 4.2.4.

A continuación se muestran superficies que representan los límites inferiores de probabilidad de respuesta correspondientes a los experimentos de la Fig. 4.7 para diferentes intervalos dentro de las ventanas de estimulación y de baseline especificadas en la tabla del apartado anterior. Los tamaños de intervalo Δ utilizados son los especificados con anterioridad.

En las superficies puede verse como, de acuerdo con lo esperado, las neuronas KCs tienen límites inferiores de probabilidad de respuesta más pequeños (no llegan a superar un valor de 0,9) debido a la poca actividad que presentan, lo que hace más difícil determinar con seguridad si se ha producido o no la respuesta, mientras que las neuronas PNs presentan límites inferiores mucho más altos debido a que su mayor actividad hace que se cuente con más datos para calcular las distribuciones y compararlas.

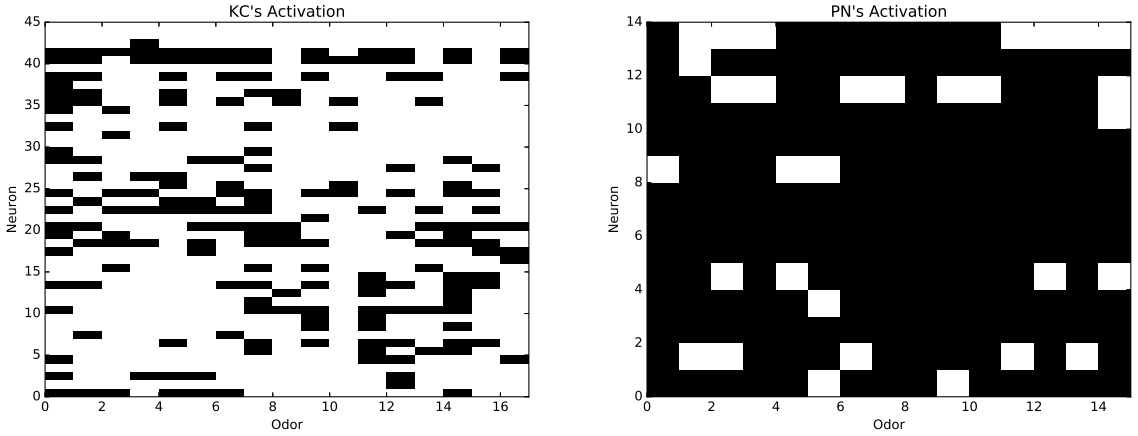
Si elegimos un umbral de probabilidad de respuesta p_r para considerar que se ha producido una respuesta que deje por debajo menos del 5 % de falsos positivos, para este límite será de



(a) (b)

(c) (d)

Figura 4.7: Superficies del límite inferior de probabilidad de respuesta en función de diversos tamaños y puntos de inicio de la ventana de estimulación para dos experimento sobre KC's y dos experimentos sobre PN's



(a) (b)

Figura 4.8: Diagramas de activación para KC's y PN's. El color negro indica que la neurona responde a un olor específico mientras el blanco corresponde a la ausencia de respuesta.

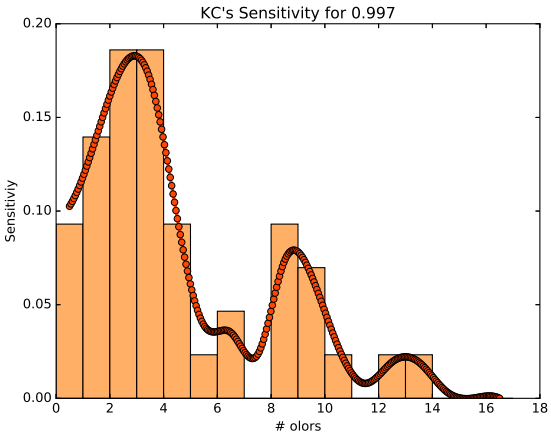
$p_{r_{KC}} = 0{,}977$ y para las PNs de $P_{r_{PN}} = 0{,}999$. En la Fig. 4.8 de activación puede observarse si se producen o no respuestas ante los diferentes estímulos por parte de todas las neuronas KCs y todas las PNs de los experimentos, comprobándose así que, mientras las PNs reaccionan ante la mayoría de estímulos, las KCs parecen mucho más selectivas, respondiendo cada una de ellas a un número muy reducido de odorantes, lo que parece sugerir que las PNs son neuronas más generalistas mientras que las KCs se encargan del reconocimiento específico de odorantes, idea que se seguirá explorando en la sección 4.2.3..

4.2.3. Sensitividad de las KCs vs. PNs

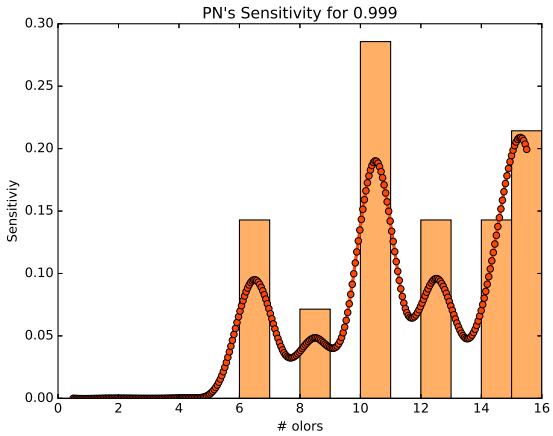
La sensitividad, definida como la probabilidad de que una neurona reaccione ante n estímulos de un total de N, es decir, $P(n|N)$ [2], puede ser un parámetro de gran utilidad a la hora de averiguar cómo se comporta un sistema de forma global ante determinados estímulos en el tiempo. A continuación se muestran los gráficos de sensitividad (Fig. 4.9) para las PNs y las KCs, suavizados mediante una Gaussiana con $\sigma = 0{,}6$, para la ventana correspondiente al intervalo de estimulación en los experimentos ($\Delta_{t=4} = 1$ en el caso de las KCs, $\Delta_{t=2} = 1$ en el caso de las PNs).

El cálculo de la sensitividad nos permite saber que son muy poco sensitivos en general y que la mayoría de ellos solo reaccionan ante dos estímulos o menos, registrándose un pequeño máximo en seis estímulos, lo que podría significar la presencia de algunas neuronas con respuesta más generalista entre la mayoría de KCs que muestran un patrón de respuesta muy específico. Por otro lado, las PNs presentan respuestas ante la mayoría de estímulos, situándose el máximo entre los once y los doce odorantes, de lo que se deduce que se trata de neuronas generalistas que captan las características más importantes de los estímulos. Esto es coherente con la forma en la que se organiza el sistema olfativo de la langosta voladora, ya que en el caso de específico de este insecto, las PNs son pluriglomerulares, es decir, reciben la información sobre los estímulos captados de diferentes receptores olfativos, mientras que en muchos otros insectos, este no es el caso [19].

Otra medida interesante que puede tomarse para analizar el comportamiento de las neuronas del sistema es representar la distribución del límite inferior de probabilidad de respuesta para todos los experimentos realizados. En la (Fig. 4.10) se muestran los histogramas de estas distribuciones para PNs y KCs en las mismas condiciones que en el caso anterior.

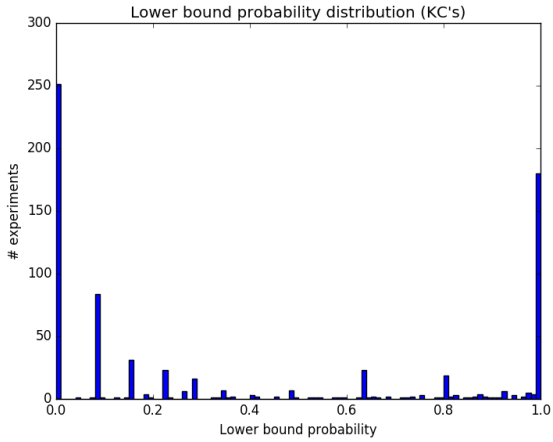


(a) Sensitividad de los KC's para $\Delta_3 = 1s$ y $p_r = 0{,}997$



(b) Sensitividad de los PN's para $\Delta_2 = 1s$ y $p_r = 0{,}999$

Figura 4.9: Sensitividad en el intervalo de estimulación para KC's y PN's



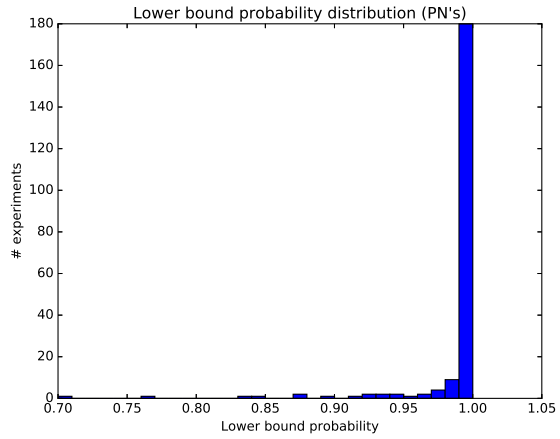


Figura 4.10: Distribuciones del límite inferior de probabilidad de respuesta para KC's y PN's

La distribución de la probabilidad de respuesta ratifica lo observado a través de la sensitividad: para las KCs, la probabilidad de respuesta está concentrada entre los valores de 0.0 y 1.0, siendo superior para el primero, mientras que en el caso de las PNs, se encuentra concentrada en el valor de 1.0, lo que es coherente, ya que se ha visto que las PNs son más generalistas y casi siempre presentan respuestas ante estímulos.

Las conclusiones que pueden extraerse de este análisis son similares a las obtenidas en [5] y [2] y a su vez son consistentes con la forma en la que se encuentra estructurado el sistema olfativo de la langosta ya que, como se explicó, en las PNs del lóbulo antenal la información se comprime para capturar los aspectos más generales de los estímulos que les llegan y proyectan dicha información en un espacio de más dimensiones en el cuerpo fungiforme formado por las KCs, de manera que cada olor pueda codificarse sin que se produzcan colisiones, lo que es coherente con el hecho de que cada KC reaccione ante muy pocos odorantes de los presentados durante los experimentos, o que ni siquiera presenten una respuesta.

4.2.4. Codificación temporal

Como se explicó en la sección 2.3., la distribución de disparos neuronales es el medio mediante el cual la información se procesa y transmite en el sistema nervioso. Una de las formas en las que esto puede realizarse es mediante la evolución de la tasa de disparos en el tiempo, con periodos en los que la tasa de estos se incrementa o la introducción de tiempos de latencia entre la captación del estímulo y el comienzo de la respuesta, lo que se denomina codificación temporal. Está demostrado que este tipo de codificación está presente en el sistema olfativo de insectos como la langosta [20, 21].

Dada la libertad que las herramientas desarrolladas dan a la hora de seleccionar ventanas de estimación para calcular medidas de sensitividad, se consideró interesante poder realizar un análisis de cómo evoluciona la sensitividad en el tiempo para observar aspectos de la codificación temporal llevada a cabo por las neuronas.

Antes de presentar los resultados de dicho análisis, se muestran varios ejemplos de codificación temporal en las neuronas PNs, en las que es especialmente clara la ocurrencia de este fenómeno. En la (Fig. 4.11) se muestran las series de disparos neuronales para el mismo odorante en todas sus diferentes concentraciones para la misma neurona:

Además de variar en función de la concentración, también se producen cambios en los patrones de disparo al presentar diferentes odorantes a la misma neurona, como se puede ver en la (Fig. 4.12). Como se ha podido observar, a medida que la concentración del odorante aumenta, la tasa de disparo aumenta también y el tiempo de latencia entre la estimulación y la respuesta se hace más amplio (este efecto se puede apreciar mejor en las figuras correspondientes al hexanol), aunque la distribución de disparos conserva la misma forma, lo que podría evidenciar que la tasa de disparo y la latencia son formas de codificar la intensidad del estímulo y la distribución de los disparos en el tiempo, un identificador del olor presentado o una representación de sus características.

Teniendo en cuenta lo expuesto anteriormente y las figuras sobre las PNs, se puede comprender mejor en qué consiste la codificación temporal. Como se dijo al comienzo de esta sección, explorando la sensitividad del sistema en diferentes ventanas temporales también puede obtenerse información sobre cómo evoluciona la respuesta del sistema en función del tiempo y, por tanto, sobre la codificación temporal. En la (Fig. A.5) se muestra la evolución de la sensitividad para diferentes ventanas de estimación.

En el caso de los KC's, se ha tomado como umbral de probabilidad $p_r = 0{,}997$ debido a que este valor reduce los falsos positivos a menos del su respuesta es corta en el tiempo ya que la

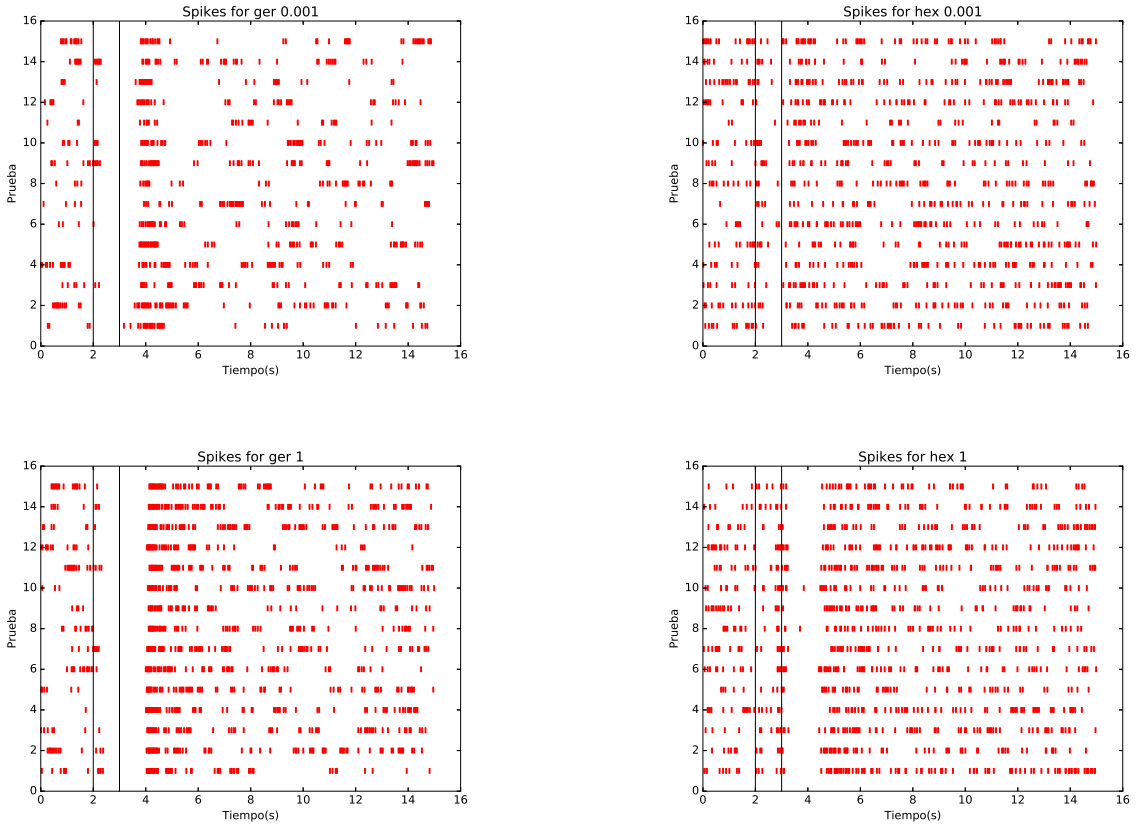


Figura 4.11: Disparos neuronales para los odorantes hexanol y geraniol a concentraciones mínima (0.001) y máxima (1)

sensitividad solo tarda 2s en normalizarse y volver a presentar una forma correspondiente al 5 % y fue el utilizado en [2]. Como puede verse, durante los intervalos situados en el baseline (primera y última figura), la mayoría de neuronas no presentan ninguna respuesta, aunque puntualmente presenta neuronas muy sensibles a estímulos 1s y 2s pasado el intervalo de estimulación. De esto se desprende que los KCs no presentan una codificación demasiado elaborada en el tiempo.

Para los PN's (Fig. A.6), la respuesta es más prolongada en el tiempo, ya que las neuronas responden a más olores después del periodo de estimulación, hasta 3s después y tarda en volver a valores parecidos a los del baseline, en los que se producen pocas respuestas, lo que es evidencia de un código temporal más elaborado que en el caso anterior. En el cálculo, se ha omitido el periodo de baseline de [0s - 2s] para poder calcular la sensitividad en el periodo [1s - 2s] de tal forma que el baseline se mantuviese constante para todas las mediciones.

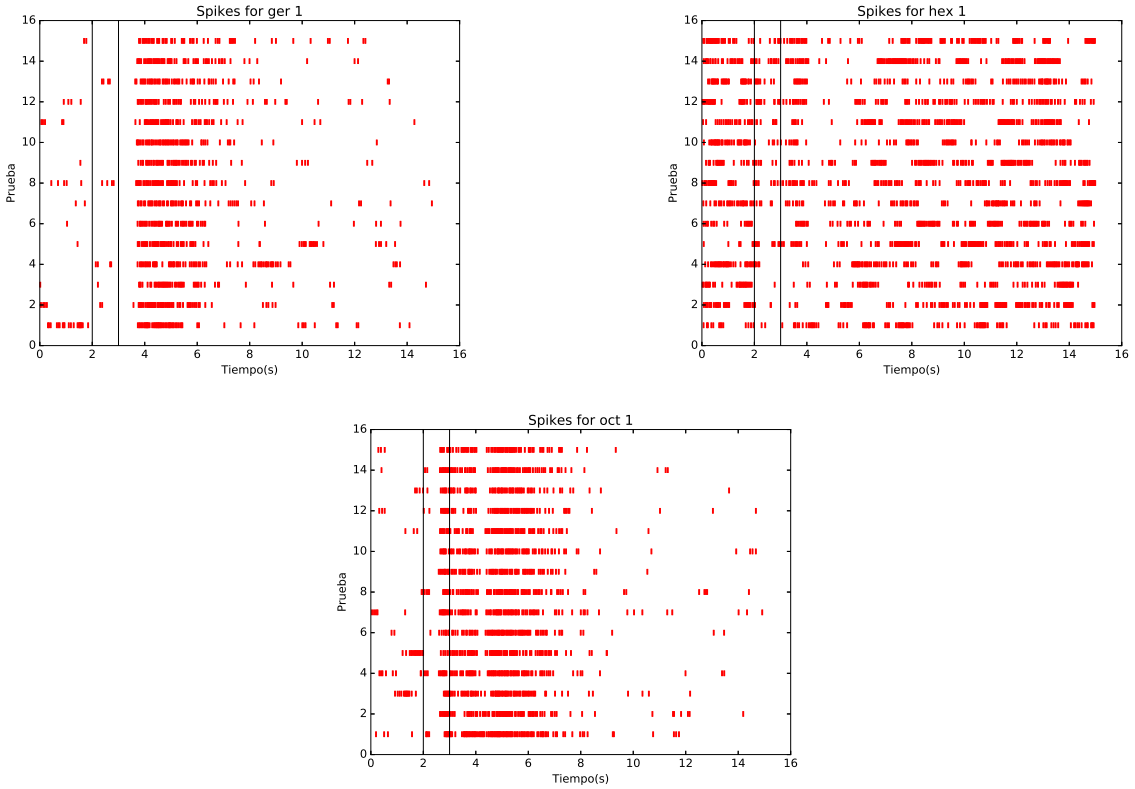


Figura 4.12: Disparos neuronales para los odorantes geraniol, hexanol y octanol a concentración máxima

5

Conclusiones y trabajo futuro

5.1. Conclusiones

El proyecto desarrollado ha cumplido con el objetivo principal que se planteó de desarrollar una metodología que permitiese detectar la respuesta de un sistema ante un estímulo en un entorno con incertidumbre. Para ello, se han seguido los siguientes pasos:

- En primer lugar se analizó el problema, diferentes técnicas de CPA que pueden aplicarse para resolverlo y se comprendió el método bayesiano propuesto por el GNB en este contexto.
- Se diseñaron e implementaron las herramientas software necesarias para la implementación del método de CPA bayesiano propuesto y otros algoritmos útiles para la reducción de falsos positivos y la selección del tamaño de histograma óptimo.
- Se realizó un modelo sencillo para la generación de datos de forma artificial, con los que se validó la metodología implementada.
- Se aplicó el método desarrollado a datos reales procedentes de las células KCs y PNs del sistema olfativo de la langosta y se llevó a cabo un análisis sobre dichos experimentos. Los resultados obtenidos están de acuerdo con las conclusiones sobre el funcionamiento de estas neuronas expuestos en [5] y [2] y se ha obtenido información adicional sobre su comportamiento, como en el caso de la codificación temporal llevada a cabo por las neuronas del sistema.

Además de lo anterior, se ha demostrado la necesidad de contar con métodos robustos y automatizados que permitan analizar los datos procedentes de los experimentos en neurociencia de una forma eficaz y confiable, así como la necesidad de definir un formato universal para la representación y etiquetado de los mismos.

5.2. Trabajo futuro

El presente proyecto puede seguir desarrollándose en el futuro desarrollando las tareas que se exponen a continuación:

- La metodología desarrollada puede ser aplicada a datos procedentes de otros experimentos de neurociencia o procedente de otros campos. En concreto, se planea su aplicación sobre datos de peces eléctricos para analizar los códigos que estos utilizan al comunicarse y datos de mediciones sobre la calidad del aire para detectar presencia.
- Integración de las herramientas software desarrolladas con una interfaz gráfica que permita que los usuarios puedan utilizarla de una forma sencilla sin necesidad de tener conocimientos sobre programación. Para ello, las herramientas desarrolladas fueron diseñadas teniendo en cuenta esta posterior integración.
- Integración de las herramientas software con una base de datos que permita almacenar los datos procedentes de los experimentos y etiquetarlos de una forma más robusta y estable.
- Publicación de las herramientas software desarrolladas para que puedan ser utilizadas por otros investigadores, lo que puede contribuir a la estandarización del tratamiento de los datos en experimentos de neurociencia.
- Comparación del método bayesiano propuesto por el GNB con otros métodos de CPA disponibles como el método Gaussiano implementado en las herramientas software desarrolladas.
- Estudiar cómo varía la probabilidad de respuesta en función del baseline y calcular el baseline óptimo basado en lo anterior.
- Continuación de la investigación sobre el sistema olfativo y cómo procesa información durante el Máster en Investigación e Innovación en TIC.

Glosario

- **GNB**: Grupo de Neurocomputación Biológica
- **UAM**: Universidad Autónoma de Madrid
- **CPA**: *Change Point Analysis*. Técnicas para detectar puntos de cambio en la distribución de eventos en una serie temporal.
- ***Change point***: punto del tiempo en una serie temporal a partir del cual se produce un cambio en la distribución de la serie temporal.
- **MISE**: Mean Integrated Squared Error. Error cuadrático medio integrado empleado para calcular la diferencia entre dos funciones.
- **Baseline**: actividad inherente a una neurona cuando no está siendo estimulada específicamente.
- **LA**: lóbulo antenal de la langosta.
- **CF**: cuerpo fungiforme del sistema olfativo de la langosta.
- **PN**: Proyection Neuron. Neuronas ubicadas en el lóbulo antenal de la langosta.
- **KC**: Kenyon Cell. Neuronas localizadas en el cuerpo fungiforme del sistema olfativo de la langosta.

Bibliografía

[1] T. Mosqueiro, M. Strube-Bloss, R. Tuma, R. Pinto, B. H. Smith, and R. Huerta. Non-parametric change point detection for spike trains. *2016 Annual Conference on Information Science and Systems*, pages 65–73, 2016.

[2] F. B. Rodríguez and R. Huerta. Techniques for temporal detection of neural sensitivity to external stimulation. *Biol Cybern*, pages 289–297, 2009.

[3] J. A. Sigüenza. *Cómo funciona el cerebro*. Eudema, Madrid, 1993.

[4] A. Longstaff. *Neuroscience*. Springer, New York, 2000.

[5] J. Pérez-Orive, O. Mazor, G. C. Turner, S. Cassenaer, R. I. Wilson, and G. Laurent. Oscillations and sparsening of odor representations in the mushroom body. *Science*, 297(5580):359–65, 2002.

[6] F. Rieke, D. Warland, R. R. Van Steveninck, and W. Bialek. *Spikes: Exploring the neuronal code*. MIT Press, Massachusetts Institue of Technology, 1997.

[7] R. Christensen. Testing Fisher, Neyman, Pearson and Bayes. *Journal of the American Statistical Association*, 59(2):121–126, 2005.

[8] J. Neyman and E. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Phil Trans R Soc Lond Ser A*, 231:289–337, 1933.

[9] M. Lavielle and G. Teyssière. Detection of multiple change-points in multivariate time series. *Lithuanian Mathematical Journal*, 46(3):287–306, 2006.

[10] B. Efron. Bootstrap methods: Another look at the Jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[11] D. S. Matteson and N. A. James. A nonparametric approach for multiple change point analysis of multivariate data. *Journal of the American Statistical Association*, 109(505):334–345, 2014.

[12] E. N. Brown R. E. Kass, V. Ventura. Statistical Issues in the Analysis of Neuronal Data. *Journal of Neurophysiology*, 94(1):8–25, 2005.

[13] D. KNuth. *The Art of Computer Programming, Vol 1: Fundamental Algorithms*. Addison-Wesley, United States, 1968-.

[14] D. Peña. *Fundamentos de Estadística*. Alianza Editorial, Madrid, 2008.

[15] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[16] S. Shinomoto. *Analysis of Parallel Spike Trains*. Springer US, United States, 2010.

[17] S. Shinomoto and H. Shimazaki. A method for selecting the bin size of a time histogram. *Neural Comput*, 19(6):1503–27, 2007.

[18] F. B. Rodríguez, R. Huerta, and M. L. Aylwin. Neural sensitivity to odorants in deprived and normal olfactory bulbs. *Plos One*, 8(4):e60745, 2013.

[19] A. Menini. *The neurobiology of olfaction*. CRC Press, 2009.

[20] F. Theunissen and J. P. Miller. Temporal encoding in nervous systems: A rigorous definition. *Journal of Computational Neuroscience*, 2(2):149–162, 1995.

[21] M. Stopfer, V. Jayaraman, and G. Laurent. Intensity versus Identity Coding in an Olfactory System. *Neuron*, 39(6):991–1004, 2003.

A

Figuras adicionales

En este apéndice se incluyen las figuras adicionales a las que se ha hecho referencia en la memoria pero que no están incluídas en el cuerpo de la misma.

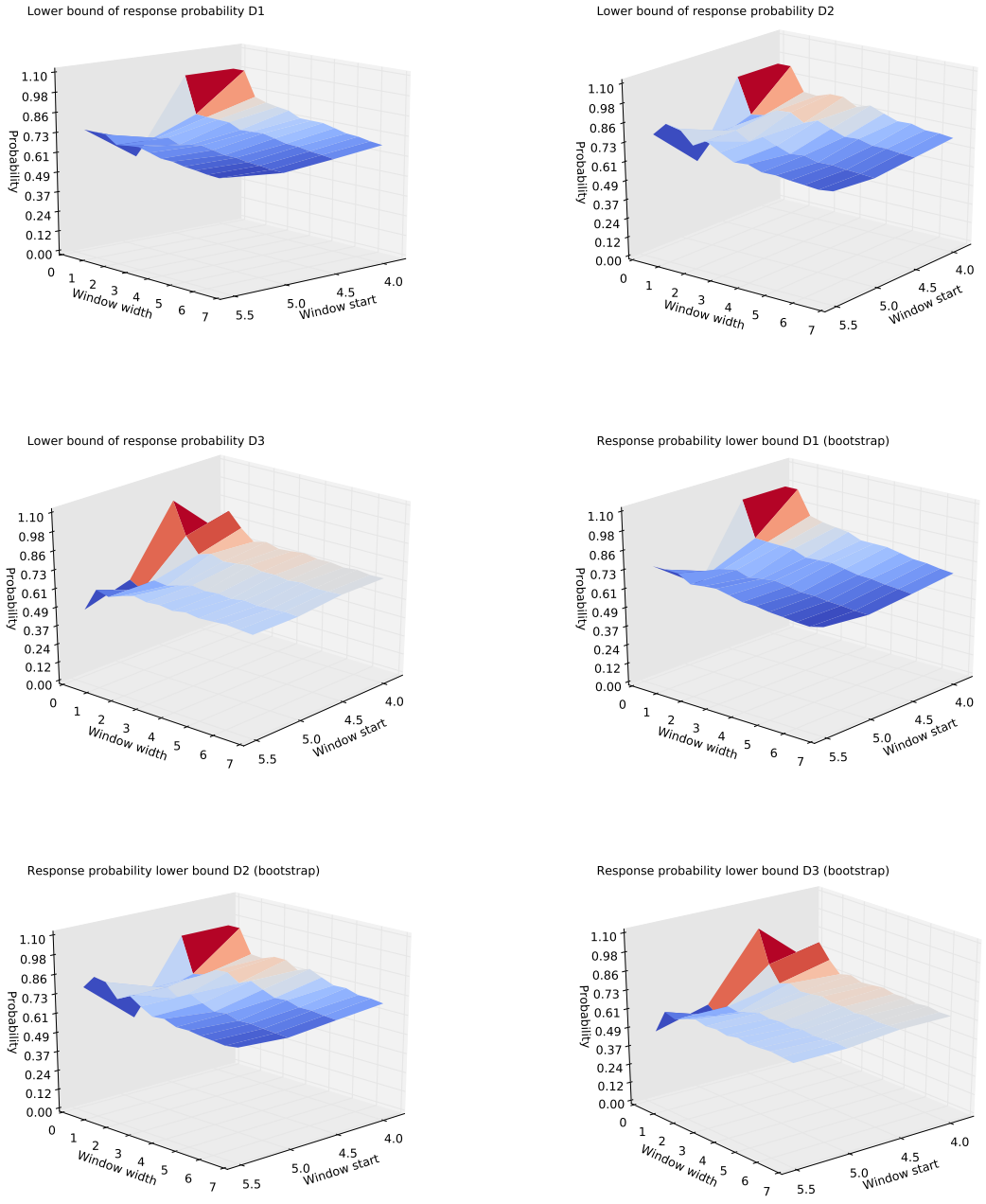


Figura A.1: Límites inferiores de probabilidad de respuesta para los datasets D_1, D_2 y D_3 en función del punto de inicio y duración de la ventana

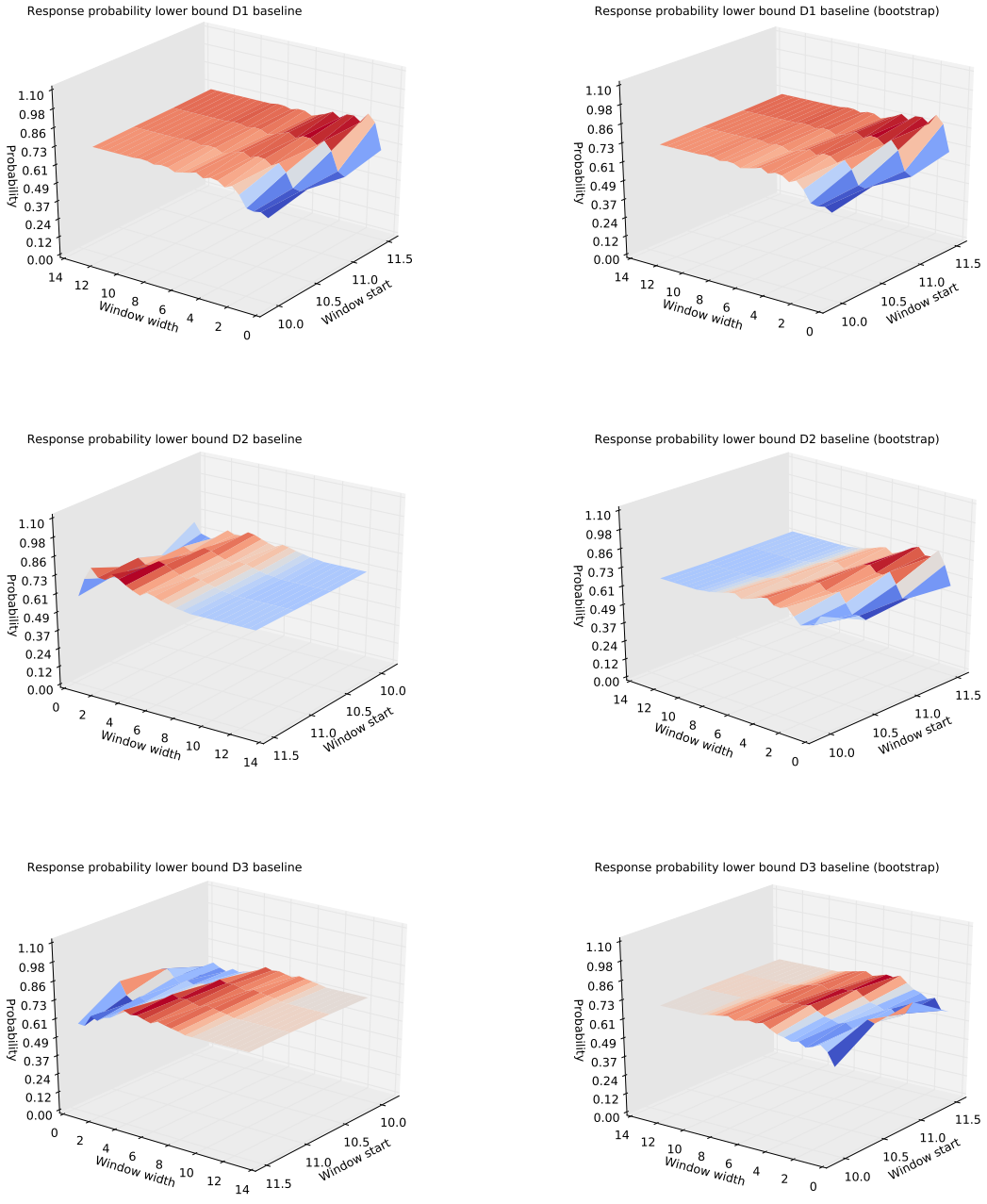


Figura A.2: Límites inferiores de probabilidad de respuesta para los datasets D_1, D_2 y D_3 en función del punto de inicio y duración de la ventana dentro del periodo de baseline

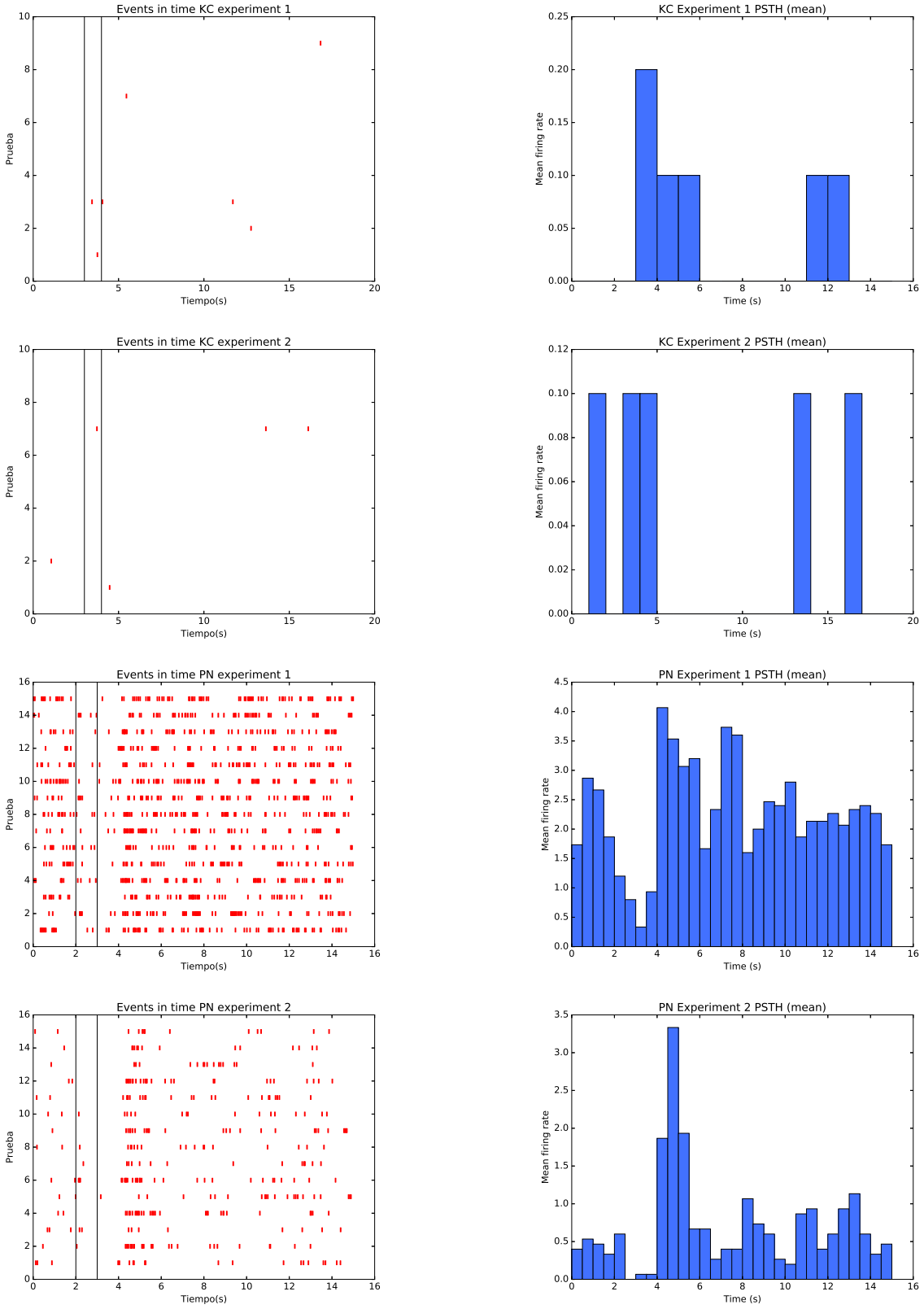
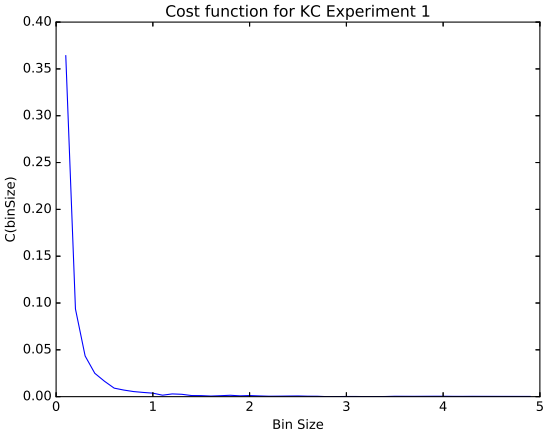
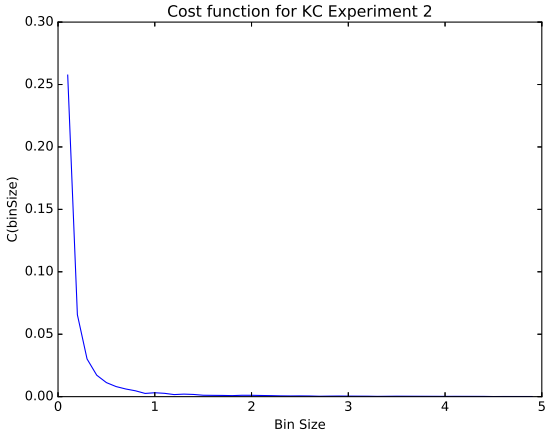


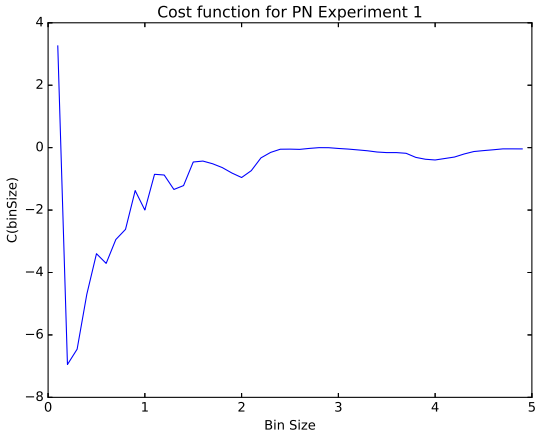
Figura A.3: Diagramas de disparos neuronales y PSTHs para cuatro experimentos realizados sobre células KC's y PN's



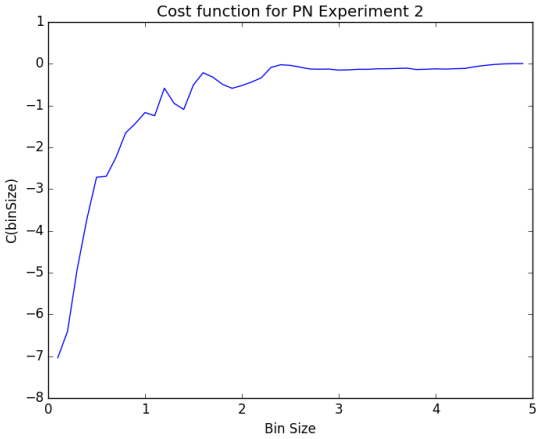
(a)



(b)

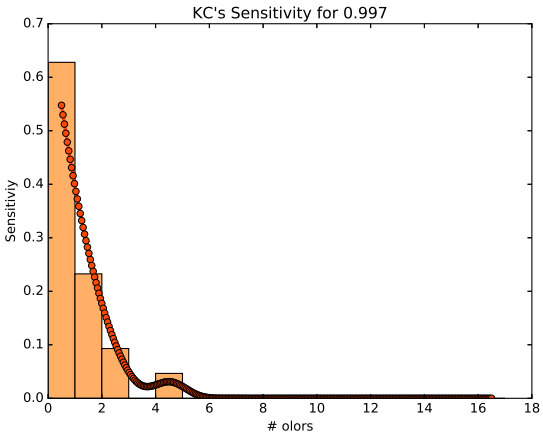


(c)

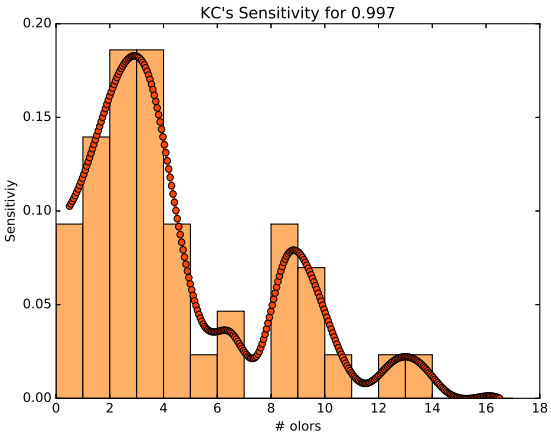


(d)

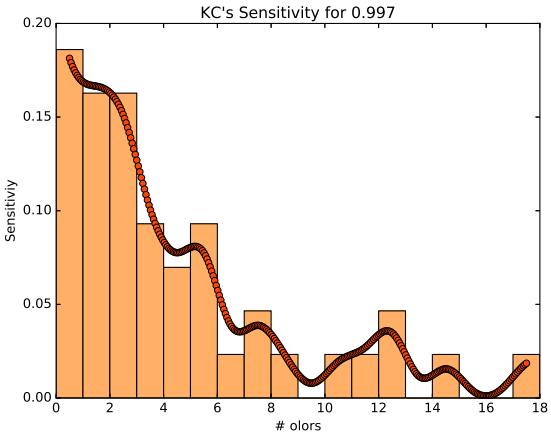
Figura A.4: $C(\Delta)$ suavizado para los experimentos seleccionados de KC's y PN's



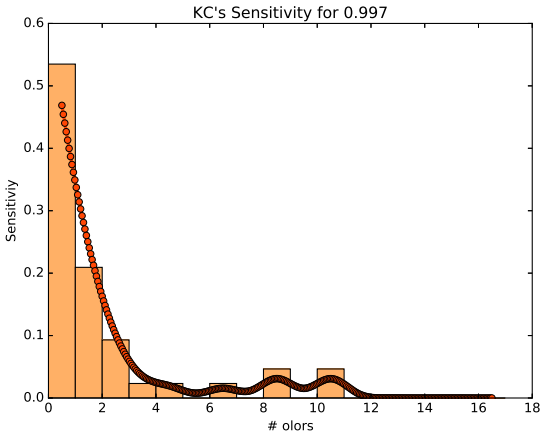
(a) Sensitividad en [2s - 3s]



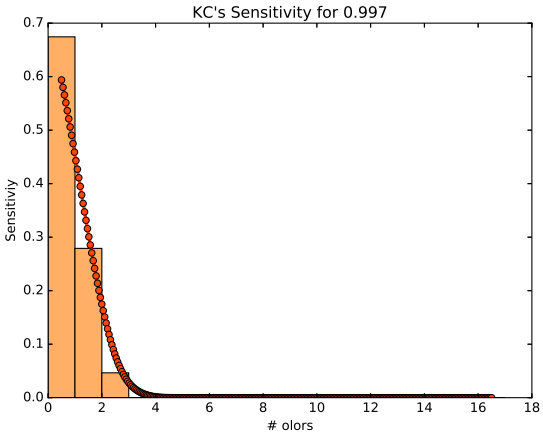
(b) Sensitividad en [3s - 4s]



(c) Sensitividad en [4s - 5s]

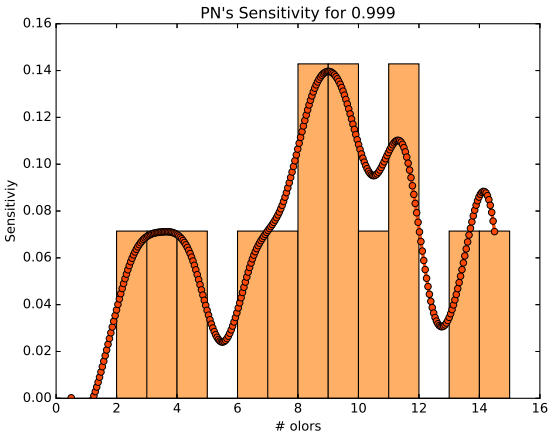


(d) Sensitividad en [5s - 6s]

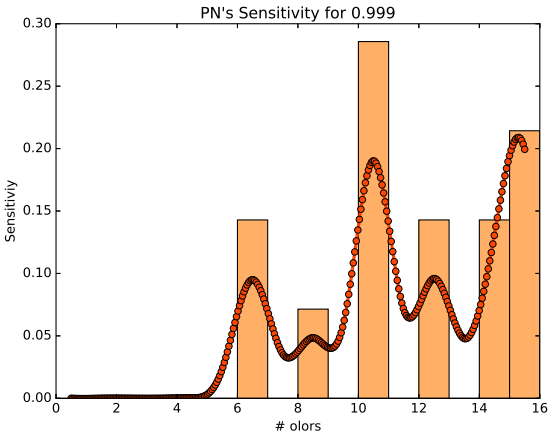


(e) Sensitividad en [7s - 8s]

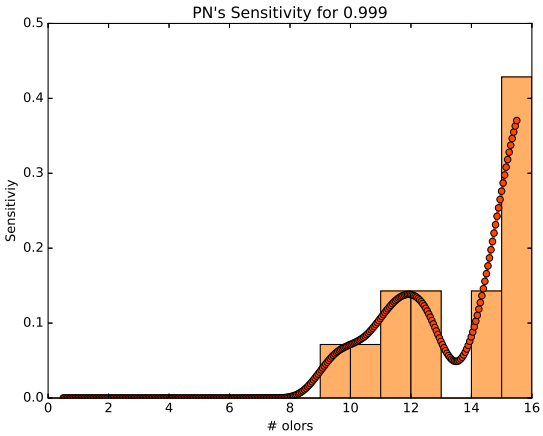
Figura A.5: Evolución de la sensitividad de las neuronas KC's en el tiempo



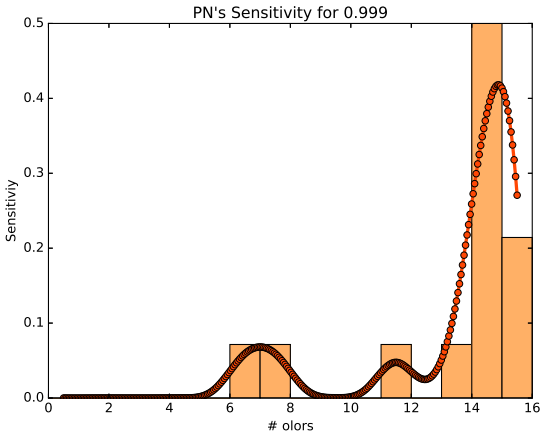
(a) Sensitividad en [1s - 2s]



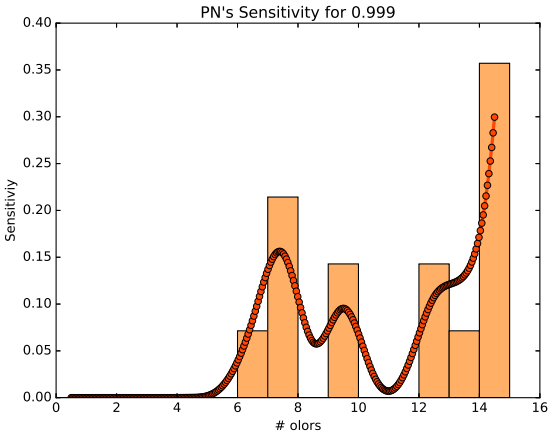
(b) Sensitividad en [2s - 3s]



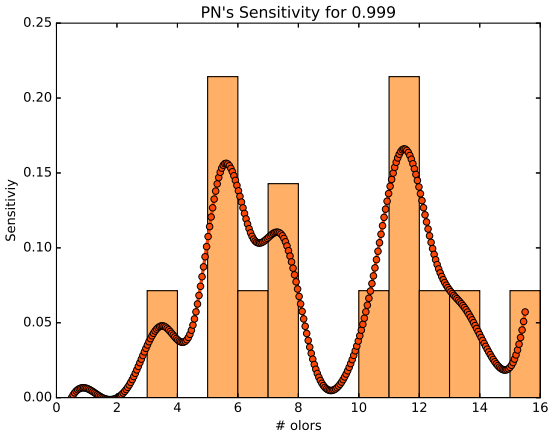
(c) Sensitividad en [3s - 4s]



(d) Sensitividad en [4s - 5s]



(e) Sensitividad en [5s - 6s]



(f) Sensitividad en [7s - 8s]

Figura A.6: Evolución de la sensitividad de las neuronas PN's en el tiempo

B

Herramientas software desarrolladas

En este anexo se describen las herramientas software que se han implementado para poder aplicar la metodología propuesta en el proyecto. Para ello, se describe el formato que deben cumplir los datos para ser cargados en el programa, una descripción de las clases y funciones implementadas y, por último, el código fuente de las herramientas.

B.1. Formato de los datos

Los datos pueden ser cargados por la clase DataLoader siguiendo dos formatos que se explican a continuación:

- Formato 1: este formato lee los datos de los experimentos como una serie de instantes del tiempo en los que se ha producido un evento. Por tanto, en el fichero de datos de un sistema (por ejemplo, de una neurona), solo se incluirán dichos tiempos, que se cuantificarán desde el inicio del experimento, sin resetear entre pruebas. Para el etiquetado de los datos, se incluirá otro fichero en el que se especifiquen los nombres de los ficheros que se quieren cargar junto a sus etiquetas.

  El formato de los ficheros que contienen datos sobre los eventos en los experimentos sería el siguiente:

  ```
  224.305624622001
  263.395403534709
  461.880854781265
  562.597271688731
          ...
  ```

  Donde cada línea representa un instante del tiempo desde el inicio del experimento en el que se ha registrado un evento.

  Por otro lado, el formato del fichero de propiedades y etiquetas sería como sigue:

  ```
  # nombre de todas las etiquetas que se van a emplear separados
      por comas
  nada,luz,repetido,olor
  ```

```
# numero total de bloques de ficheros
13
# comienzo de un bloque
# numero de experimentos en cada fichero
19
# nombres de los ficheros de datos pertenecientes al bloque con
    el formato descrito arriba que comparten un mismo etiquetado
L72s1_t12_C1.dat
L72s1_t12_C2.dat
L72s1_t12_C3.dat
# cada etiqueta se asocia a los indices de los experimentos
    correspondientes
nada,1
repetido,19
olor,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18

# tras la ultima etiqueta, comienza un nuevo bloque
```

- Formato 2: este formato contiene todos los experimentos realizados sobre el mismo sistema en un solo fichero, que contiene los tiempos en los que se produjeron los eventos en tiempo relativo al inicio de cada prueba y las etiquetas correspondientes a cada experimento. No son necesarios más ficheros adicionales. A continuación se muestra un ejemplo de este formato:

```
# caracteristicas de los experimentos
# numero de pruebas en cada experimento
15
# numero de experimentos
15
# identificador del conjunto de experimentos
neuron1
# etiquetas posibles que se van a utilizar
ger hex oct 0.001 0.01 0.05 0.1 1
# duracion de una prueba en segundos
20
# comienzo del primer experimento
# numero total de eventos en el experimento
615
# etiquetas del experimento
ger 0.001
# instantes del tiempo en los que se registran eventos con el
    formato 'numero de prueba | instante de tiempo'
1.0 0.6679
1.0 1.7807
1.0 2.0603
1.0 2.1282
    ...
#comienzo del siguiente experimento
```

Además de lo anterior, también está permitido cargar experimentos individuales en el programa siguiendo el Formato 2.

B.2. Descripción de las herramientas desarrolladas

A continuación se describe cada una de las clases implementadas y los métodos que contiene. En el código adjunto puede consultarse la documentación relativa a la funcionalidad implementada por cada método, así como sus parámetros y sus valores de retorno.

- **DataLoader:** en esta clase se encuentran los métodos necesarios para procesar los ficheros con los datos de los experimentos y almacenarlos en estructuras que puedan ser manejadas por el resto de herramientas. Todos los experimentos correspondientes a un mismo sistema se almacenan en un mismo ExperimentSet, que a su vez contiene los objetos Experimento correspondientes. Los métodos de esta clase se listan a continuación:
  - loadDataAbsoluteFormat()
  - loadDataRelativeFormat()
  - processBlock()
  - loadFileAbsolute()
  - loadFileIndividual()
  - loadFileRelative()
- **Experiment:** contiene los datos sobre los eventos de cada uno de los experimentos para las diferentes pruebas realizadas en forma de matrices.
- **ExperimentSet:** contiene varios experimentos agrupados con unas etiquetas y características comunes. Solo contiene la función getExperiments() que permite filtrar experimentos por etiquetas.
- **DataAnalyzer:** esta clase contiene los métodos necesarios para realizar el análisis individual de cada uno de los experimentos en un ExperimentSet, así como para calcular probabilidades de respuesta y proporcionar a otras herramientas la información que necesitan.
  - precalculateProbs()
  - getBinResponses()
  - testResponse()
  - calculateResponseProbs()
  - getProbBound()
  - calculateExperimentResponseProbs()
  - responseProbWindowWidth()
  - responseProbWindowStart()
  - responseProbWStartWidth()
- **Sensitivity:** contiene los métodos necesarios para obtener el análisis global de todos los experimentos sobre el sistema y su sensitividad.
  - dataAnalyzers()
  - calculateResponses()
  - getProbBoundData()
  - getBestProbabilityBound()
  - getSensivity()

- **TestResponseMethod:** interfaz que deben implementar todos los métodos de CPA que se añadan al programa, de forma que puedan utilizarse de forma opaca al resto de las herramientas.
  - getResponseProb()
  - calculateBinDistribution()
  - calculateBinsResponseProb()
- **BayesianTest:** contiene el método de estimación de la probabilidad de respuesta bayesiano descrito en los apartados anteriores e implementa la interfaz TestResponseMethod.
  - Métodos de la interfaz TestResponseMethod
  - testResponseProb()
  - calculateBinsResponseProb()
  - calculateProbDistribution()
  - independentResponseProb()
- **BayesianTestBootstrap:** variación del método anterior en el que se utiliza bootstrapping para mejorar la estimación de las distribuciones de probabilidad de los eventos. Sobreescribe al método calculateBinDistribution() de BayesianTest.
- **GaussianMethod:** implementa el método de CPA basado en Gauss mencionado anteriormente para poder establecer una comparativa con la alternativa del bayesiano.
  - Métodos de la interfaz TestResponseMethod
  - costFunction()
  - getBestBinSize()
  - testResponseN()
- **Plotter:** módulo auxiliar que contiene las funciones necesarias para realizar las representaciones gráficas de los datos.
  - plotBinsResponseProb()
  - plotRaster()
  - plotSurface()
  - plotFalsePositives()
  - plotXY()
  - getSmoothGaussianVal()
  - plotSensitivity()
  - plotPSTHD()
  - activationPlot()
  - plotCostFunction()
  - show()

B.3. Código

B.3.1. DataLoader

```
class DataLoader(object):
    """
    This class loads the data of the experiment files.
    The idea is to open a directory, gets all files in directory and
       loads the data
    in one experimentSet per neuron.
    """

    def __init__(self):
        """
        Empty constructor
        """
        pass

    def loadDataRelativeFormat(self, dataDir):
        """ Loads data from one directory, opening all files. The
           files must
        follow the relative format, in which event times are counted
           from the
        beginning of the trial and labels must be specified within
           the same document.
        :dataDir: directory where data event files are
        :experimentTime: total time of every trial
        :nTrials: number of trials
        """
        experimentSet = []
        onlyfiles = [f for f in listdir(dataDir) if isfile(join(
           dataDir, f))]
        for f in onlyfiles:
            experimentSet.append(self.loadFileRelative(dataDir + f))

        return experimentSet


    def loadDataAbsoluteFormat(self, dataDir, propertiesFile,
       trialTime, nTrials):
        """ Loads data from one directory, opening all files. The
           files must
        follow the absolute format, in which event times are counted
           from the
        beginning of the experiment and labels and filenames must be
           specified in a differente document.
        :dataDir: directory where data event files are
        :propertiesFile: filepath where the labels and names files is
        :trialTime: total time for each trial
        :nTrials: number of trials in the document
```

```
        """
        experimentSets = []


        with open(propertiesFile, 'r') as f:
            labels = f.readline().strip().split()
            nSets = int(f.readline().strip())
            for i in range(0, nSets):
                experimentSets.extend(self.processBlock(f, labels,
                    dataDir, trialTime, nTrials))
        return experimentSets


    def processBlock(self, f, labels, dataDir, experimentTime,
        nTrials):
        """
        Process a block within the properties file, that contains
            several data file names
        and the labels associated to them.
        :f: properties file
        :dataDir: directory where the data files are located
        :experimentTime: trial total time
        :nTrials: number of trials in every experiment
        :return: ExperimentSet containing the Experiments read from
            dataDir
        """

        nExperiments = int(f.readline().strip())
        line = f.readline().strip()

        experimentSets = {}

        while "." in line:
            experiments = np.array([])

            experimentsD = self.loadFileAbsolute(dataDir + line,
                nExperiments, experimentTime, nTrials)

            for key in experimentsD:
                experiments = np.append(experiments, Experiment.
                    Experiment(experimentsD[key], nTrials, key))

            experimentSets[line] = experiments

            line = f.readline().strip()

        properties = {}

        while line is not "":
            props = line.split(",")
            properties[props[0]] = np.array([x for x in props[1:]])
```

```
            properties[props[0]] = properties[props[0]].astype(int)
            line = f.readline().strip()

        experimentSet = []
        for key in experimentSets:
            experimentSet.append(Experiment.ExperimentSet(key,
                properties, experimentSets[key], experimentTime, len(
                experimentSets[key])))
        return experimentSet


    def loadFileAbsolute(self, filePath, nExperiments, experimentTime
        , nTrials):
        """
        Builds Experiment objects from the event data file read from
            a data file and
        returns them. The file must follow the absolute format
        :filePath: path where the data files are
        :nExperiments: number of experiments to be read
        :experimentTime: time of each trial in the experiment
        :nTrials: number of trials in experiment
        :return: dictionary containing the Experiment objects
            corresponding to the file read
        """
        experiments = {}

        for i in range(nExperiments):
            experiments[i] = np.ndarray((0,2))

        totalTime = experimentTime*nTrials
        with open(filePath, 'r') as f:
            lines = f.readlines()

            for line in lines:
                eventTime = float(line.strip())
                nExperiment = int(eventTime/totalTime)
                eventTime = eventTime - nExperiment * totalTime
                nTrial = int(eventTime/experimentTime)
                eventTime = eventTime - nTrial * experimentTime

                eventList = experiments.get(nExperiment)

                eventList = np.vstack((eventList, (float(nTrial) + 1,
                    eventTime)))

                experiments[nExperiment] = eventList

        return experiments


    def loadFileIndividual(self, filePath, experimentTime, nTrials):
```

```
        """
        This method allows to load a single experiment file into the
           program, following
        the same event data format that in the relative format
        :filePath: path of the file
        :nExperiments: number of experiments that contains the file
        :experimentTime: time of each trial in the experiment
        :nTrials: number of trials in the experiment
        :return: the Experiment loaded
        """

        with open(filePath, 'r') as f:
            lines = f.readlines()
            experiments = np.array(([[x.strip() for x in line.split('
               ')] for line in lines]))
            experiments = experiments.astype(float)
            return Experiment.Experiment(experiments, nTrials,
               filePath)

    def loadFileRelative(self, filePath):
        """
        Opens an experiment file following relative format and loads
           the content into experiment/experimentSet
        structures

        :filePath: String experiment file path
        :return: an ExperimentSet containing the experiment info,
           labels and Experiment
        objects for each file
        """

        experiments = np.array([])
        with open(filePath, 'r') as f:

            # reading/processing header
            # reads number of experiments in file
            nExperiments = int(f.readline().strip())
            # reads number of trials per experiment
            nTrials = int(f.readline().strip())
            # reads identifier of experiment set
            experimentSetId = f.readline().strip()
            # reads experiments labels
            labels = f.readline().strip().split()
            properties = {}
            for label in labels:
                properties[label] = np.array([])

            # reads total duration of experiments
            totalTime = float (f.readline().strip())

            stimuliTimes = []
```

```
            # reads rest of lines in document
            lines = f.readlines()
        n = 0
        # reads the info for every experiment in document
        for i in range(nExperiments):
            # gets id of experiment
            experimentId = lines[n].strip()
            # gets number of events (spikes) in experiment
            nEvents = int(lines[n + 1].strip())
            labels = lines[n + 2].strip().split()
            for label in labels:
                ids = properties.get(label)
                ids = np.append(ids,experimentId)
                properties[label] = ids
            # loads the data into numpy matrix. Column 0: number of
                trial ; Column 1: time when spike was detected
            data = np.array(([[x.strip() for x in line.split('␣')]
                for line in lines[n + 3 : n + 3 + nEvents]]))
            data = data.astype(float)
            n = n + 3 + nEvents
            experiments = np.append(experiments, Experiment.
                Experiment(data, nTrials, experimentId))

        # returns an ExperimentSet with the info and Experiments read
        return Experiment.ExperimentSet(experimentSetId, properties,
            experiments, totalTime, nTrials)
```

B.3.2. Experiment

```
class Experiment(object):
    """
    class to store an experiment data
    """
    def __init__(self, data, nTrials, experimentId):
        """
        Constructor
        :data: float[][][] experiment data
        :nTrials: float number of trials in the experiment
        :experimentId: String id of the experiment
        :return: new experiment
        """
        self.data = data
        self.nTrials = nTrials
        self.experimentId = experimentId
```

B.3.3. ExperimentSet

```
class ExperimentSet(object):
    """
    Class that stores different experiments belonging to the
```

```
    same system
    """
    def __init__(self, setId, properties, experiments, totalTime,
        nExperiments):
        """
        Constructor
        :setId: string id of the set
        :properties: dictionary that maps labels to experiment
            indexes
        :experiments: array of Experiment objects
        :totalTime: float time for each trial
        :nExperiments: int number of experiments that the set
            contains
        :return: new Experiment Set
        """
        self.setId = setId
        self.properties = properties
        self.experiments = experiments
        self.totalTime = totalTime
        self.nExperiments = nExperiments

    def getExperiments(self, label=None):
        """
        Gets the experiments in the set that have a certain label
        :label: string[] label by which the experiments are going to
            be filtrated
        :return: Experiment[] experiments that fulfill the label
            restrictions
        """
        if label == None:
            return self.experiments
        else:
            indexes = np.array(range(self.nExperiments + 1))

            for l in label:
                # for each label, the indexes chosen are intersected
                    to perform
                # and AND operation on the labels
                labelIndexes = self.properties[l]
                indexes = np.intersect1d(indexes, labelIndexes.astype
                    (int))

            indexes = indexes.astype(int)

            return self.experiments[indexes - 1]
```

B.3.4. DataAnalyzer

```
class DataAnalyzer(object):
    """
    Class to analyze experiments individually or inside an
        experimentSet
```

```
    """
    def __init__(self, experimentSet, testResponseMethod,
        baselineIntervals, maxSpikes=20):
        """
        Constructor
        :experimentSet: experimentSet linked to this analyzers
        :testResponseMethod: test method to be used in the
            calculation of responses
        :maxSpikes: max number of spikes for bin
        """
        self.experimentSet = experimentSet
        self.maxSpikes = maxSpikes
        self.testMethods = self.precalculateProbs(testResponseMethod,
            baselineIntervals)

    def precalculateProbs(self, testResponseMethod, baselineIntervals
        ):
        """
        Creates a test response method associated with each
            experiment in the set and
        precalculates the bin distributions and response
            probabilities given the baseline
        intervals
        :testResponseMethod: test response method to be used
        :baselineIntervals: baseline intervals
        :return: dictionary that maps each experiment with a test
            method
        """
        testMethods = {}
        for experiment in self.experimentSet.experiments:
            testMethod = deepcopy(testResponseMethod)
            testMethod.setExperiment(experiment)
            testMethod.calculateBinDistribution(self.experimentSet.
                totalTime, self.maxSpikes)
            testMethod.calculateBinsResponseProb(baselineIntervals)

            testMethods[experiment] = testMethod
        return testMethods


    def getBinResponses(self, intervals):
        """
        Return the response probability of each bin within intervals
        :intervals: interval containing the bins to return their
            response probability
        :return: response probabilities of the bins in interval
        """
        binResponses = np.array([])
        for experiment in self.experimentSet.experiments:
            binResponses = np.append(binResponses, self.testMethods[
                experiment].getResponseProbs(intervals))
```

```
        return binResponses


    @staticmethod
    def testResponse(responses, probBound):
        """
        For all experiments, returns an array with 0.0 when it doesn't present a response
        and 1 when it does. It is considered a response when the response probability of the experiment
        is above probBound
        :responses: response probability of experiments
        :probBound: p_r response bound
        :return: array of 1's and 0's indicating wether there's a response or not
        """
        vals = responses >= probBound
        return vals.astype(float), np.where(vals == 1.0)[0].shape[0]


    def calculateResponseProbs(self, stimuliIntervals, baselineIntervals, label=None):
        """
        Calculates the response probability of an interval averaging the response probability
        of the bins in it.
        :stimuliInterval: stimation intervals
        :baselineIntervals: baseline intervals
        :label: labels of the experiments to be selected
        :return: the responses of each interval in stimuliIntervals
        """
        experiments = self.experimentSet.getExperiments(label)
        responses = np.ndarray((len(experiments), len(stimuliIntervals)))
        for i,experiment in enumerate(experiments):
            responses[i, :] = self.calculateExperimentResponseProbs(stimuliIntervals, experiment, baselineIntervals)
        return responses


    def getProbBound(self, experiment, falsePositives):
        """
        Gets best probability bound p_r to minimize false positives to a certain
        fraction.
        :experiment: experiment which falsePositives have to be reduced
        :falsePositives: fraction of false positives
        :return: best probBound and number of false positives for each p_r tested
```

```
        """
        testMethod = self.testMethods[experiment]
        responseProbs = testMethod.getResponseProbs()
        probBound = np.amin(responseProbs)
        probBoundFalses =  np.ndarray((0,2))

        while np.where(responseProbs > probBound)[0].shape[0]/float(
            responseProbs.shape[0]) >= falsePositives:
            newProbBound = np.amin(responseProbs[np.where(
                responseProbs > probBound)])
            probBoundFalses = np.vstack((probBoundFalses, (
                newProbBound, np.where(responseProbs > probBound)[0].
                shape[0]/float(responseProbs.shape[0]))))
            if newProbBound == probBound:
                return probBound
            else:
                probBound = newProbBound

        return probBound, probBoundFalses


    def calculateExperimentResponseProbs(self, stimuliIntervals,
        experiment, baselineIntervals):
        """
        Calculates the response probability of an interval averaging
            the response probability
        of the bins in it. Single experiment version (not necessary
            to have an ExperimentSet)
        :stimuliInterval: stimation intervals
        :baselineIntervals: baseline intervals
        :label: labels of the experiments to be selected
        :return: the responses of each interval in stimuliIntervals
        """
        data = experiment.data
        if data.size == 0:
            return 0.0

        testMethod = self.testMethods[experiment]
        prob = testMethod.getResponseProb(stimuliIntervals,
            baselineIntervals)

        return prob


    def responseProbWindowWidth(self, windowStart, maxWindowEnd,
        widthStep, experiment, baselineIntervals):
        """
        Calculates the response probability of an experiment for
            different window widths
        :windowStart: minimum start point of the window
        :maxWindowEnd: max end point of the window
```

```
        :widthStep: window width is increased the value of widthStep
            for each trial
        :experiment: the experiment tested
        :baselineIntervals: baselineIntervals
        :returns: intervals within the parameters specified, response
            probabilities and plot
        """
        intervalsStart = np.arange(windowStart, windowStart +
            maxWindowEnd, step=widthStep)
        intervalsEnd = intervalsStart + widthStep
        intervals = np.ndarray((intervalsEnd.shape[0], 2))

        intervals[:, 0] = windowStart
        intervals[:, 1] = intervalsEnd

        responseProbs = self.calculateExperimentResponseProbs(
            intervals, experiment, baselineIntervals)

        f = Plotter.plotXY("Response probability/Window Width
            starting at " + str(windowStart), "Window width (s)", "P(
            response)", intervalsEnd - windowStart, responseProbs)

        return intervalsEnd - windowStart, responseProbs, f


    def responseProbWindowStart(self, windowStart, maxWindowStart,
        windowWidth, step, experiment, baselineIntervals):
        """
        Calculates the response probability of an experiment for
            different window start points and fixed width
        :windowStart: min start point of the window
        :maxWindowStart: max start point of the window
        :windowWidth: width of the window
        :experiment: the experiment tested
        :baselineIntervals: baselineIntervals
        :returns: intervals within the parameters specified, response
            probabilities and plot
        """

        intervalsStart = np.arange(windowStart, maxWindowStart, step=
            step)
        intervalsEnd = intervalsStart + windowWidth
        intervals = np.ndarray((intervalsEnd.shape[0], 2))
        intervals[:, 0] = intervalsStart
        intervals[:, 1] = intervalsEnd

        responseProbs = self.calculateExperimentResponseProbs(
            intervals, experiment, baselineIntervals)
        f = Plotter.plotXY("Response probability/Window Start", "
            Window start (s)", "P(response)", intervalsStart,
            responseProbs)
```

```
        return intervals, responseProbs, f


    def responseProbWStartWidth(self, windowStart, maxWindowStart,
        maxWindowEnd, step, widthStep, experiment, baselineIntervals):
        """
        Calculates the response probability of an experiment for
            different window start points and widths
        :windowStart: min start point of the window
        :maxWindowStart: max start point of the window
        :maxWindowEnd: max end point of the window
        :step: step at which the start point is increased
        :widthStep: step at which window width is increased
        :experiment: the experiment tested
        :baselineIntervals: baselineIntervals
        :returns: intervals within the parameters specified, response
            probabilities and surface plot
        """
        intervalsStart = np.arange(windowStart, maxWindowStart, step=
            step)
        responses = np.ndarray((intervalsStart.shape[0], int(ceil(
            float(maxWindowEnd)/widthStep))))

        for i,interval in enumerate(intervalsStart):
            intervals, responseProbs, f = self.
                responseProbWindowWidth(interval, maxWindowEnd,
                widthStep, experiment, baselineIntervals)
            responses[i, :] = responseProbs

        Plotter.plotSurface(intervalsStart, intervals, responses.
            transpose())
        return intervalsStart, intervals, responses
```

B.3.5. Sensitivity

```
class Sensitivity(object):
    """
    Analyzes all the ExperimentSets loaded in the system to get
        information
    about sensitivity or the global p_r that is better to use
    """
    def __init__(self, experimentSets, nStimuli, testMethod,
        baselineIntervals, maxSpikes=20):
        """
        Constructor
        :experimentSets: array[] experimentSets that are going to be
            analyzed
        :nStimuli: number of stimulis in the experiments
        :testMethod: test method that is going to be used in order to
            find the response
        probabilities
```

```
        :maxSpikes: max number of spikes in each bin
        """
        self.baselineIntervals = baselineIntervals
        self.sets = experimentSets
        self.nSets = len(experimentSets)
        self.nStimuli = nStimuli
        self.testMethod = testMethod
        self.maxSpikes = maxSpikes
        self.analyzers = self.dataAnalyzers()


    def dataAnalyzers(self):
        """
        Initializes DataAnalyzers for each experimentSet in the
           system and precalculates
        all the data necessary to get the responses
        :return: dictionary mapping each experimentSet to an
           DataAnalyzer
        """
        analyzers = {}
        for s in self.sets:
            analyzer = DataAnalyzer.DataAnalyzer(s, self.testMethod,
               self.baselineIntervals, self.maxSpikes)
            analyzers[s] = analyzer
        return analyzers


    def calculateResponses(self, stimuliInterval, label=None):
        """
        Calculates the response probabilities for each interval in
           stimuliIntervals comparing
        with the baseline intervals ones. Plots the response
           probability distribution.
        :stimuliIntervals: stimation windows that are going to be
           tested for response
        :baselineIntervals: baseline intervals
        :label: labels of the experiments to be analyzed
        :return: response probabilities, its maximum and minimum
           values
        """
        self.responses = []
        maxVals = []
        minVals = []
        responses = np.array([])

        for expSet in self.sets:
            analyzer = self.analyzers[expSet]
            setRes = analyzer.calculateResponseProbs(stimuliInterval,
                self.baselineIntervals, label)
            responses = np.append(responses, setRes)
            maxVals.append(max(setRes))
```

```
            minVals.append(min(setRes))
            self.responses.append(setRes)
        plt.hist(responses, bins=np.arange(min(minVals), max(maxVals)
            + 0.01, 0.01))
        plt.title("Lower bound probability distribution")
        plt.xlabel("Lower bound probability")
        plt.ylabel("# experiments")

        return responses, max(maxVals), min(minVals)



    def getProbBoundData(self, label=None):
        """
        Calcultes the number of false positives for each value of p_r
            tested
        :baselineIntervals: baseline intervals
        :label: labels of the experiments to be analyzed
        :return: falsePositives associated to p_r bounds
        """
        responseProbs = np.array([])
        for s in self.sets:
            analyzer = self.analyzers[s]
            responseProbs = np.append(responseProbs, analyzer.
                getBinResponses(self.baselineIntervals))
        bounds = np.unique(responseProbs)
        falsePositives = np.array([])

        for prob in bounds:
            falsePositives = np.append(falsePositives, np.where(
                responseProbs > prob)[0].shape[0]/float(responseProbs.
                shape[0]))

        Plotter.plotXY("False positives", "Probability Bound", "%%
            false positives", bounds, falsePositives)
        return falsePositives, bounds


    def getBestProbabilityBound(self, label=None, falsePositives
        =0.05):
        """
        Gets the best p_r bound for all experiments in the system
        :baselineIntervals: baseline intervals
        :label: labels of the experiments to be analyzed
        :falsePositives: percentage of false positives
        :return: best p_r bound to minimize the false positives to a
            value lower than falsePositives
        """
        responseProbs = np.array([])
        for s in self.sets:
            analyzer = self.analyzers[s]
```

```
            responseProbs = np.append(responseProbs, analyzer.
                getBinResponses(self.baselineIntervals))

        probBound = np.amin(responseProbs)

        while np.where(responseProbs > probBound)[0].shape[0]/float(
            responseProbs.shape[0]) > falsePositives:
            newProbBound = np.amin(responseProbs[np.where(
                responseProbs > probBound)])

            if newProbBound == probBound:
                return probBound
            else:
                probBound = newProbBound
        return probBound


    def getSensivity(self, probBound, maxOdors=10):
        """
        Gets the sensitivity of a system given a p_r bound
        :probBound: prob bound that determines wether there's
            response or not
        for a certain response probability value
        :maxOdors: maximum number of stimuli
        :return: sensitivity probs and responses of neurons to
            stimuli
        """
        nOlors = np.array([])
        responses = np.ndarray((len(self.responses), maxOdors))
        for i,expResponse in enumerate(self.responses):
            response, nResults = DataAnalyzer.DataAnalyzer.
                testResponse(expResponse, probBound)
            responses[i, :] = np.transpose(response)
            nOlors = np.append(nOlors, nResults)
        probs = np.bincount(nOlors.astype(int))/(float(nOlors.shape
            [0]))# * self.nStimuli)

        if(probs.shape[0] < maxOdors):
            probs = np.append(probs, np.zeros(maxOdors - probs.shape
                [0]))

        Plotter.plotSensitivity(np.arange(probs.shape[0]), probs,
            probBound, 0.6)

        return probs, responses



\subsection{TestResponseMethod}\label{TestMethod}
\begin{lstlisting}[language=Python]
class TestResponseMethod(object):
```

```
    """
    Abstract class that defines the methods to be implemented by the
    rest of classes working with probability distributions
    """
    __metaclass__ = ABCMeta

    def __init__(self):
        """
        Empty constructor
        """
        pass

    def getResponseProbs(self, intervals=None):
        return self.responseProbs

    @abstractmethod
    def getResponseProb(self, stimuliIntervals, baseLineIntervals):
        """
        Returns response probability for a given intervals
        """
        pass

    @abstractmethod
    def calculateBinDistribution(self, totalTime, maxSpikes=20):
        pass

    @abstractmethod
    def calculateBinsResponseProb(self, baselineIntervals):
        pass
```

B.3.6. BayesianTest

```
class RawBayesianMethod(TestResponseMethod.TestResponseMethod):
    """
    Class that implements the bayesian test for detecting neural
       responses estimating
    the probability distribution of events directly from the rasters
    """
    def __init__(self, binSize, step, experiment):
        """
        Constructor of the class
        :binSize: float binSize
        :step: float distance between bin start points
        :experiment: experiment to be testedBinProb
        """
        self.binSize = binSize
        self.experiment = experiment
        self.step = step

    def getResponseProbs(self, intervals=None):
            if intervals == None:
                return self.responseProbs
```

```
            else:
                responseProbs = []
                for interval in intervals:
                    indexes = np.where(np.logical_and(self.intervals
                        >= interval[0], self.intervals + self.binSize
                        <= interval[1]))
                    responseProbs = np.append(responseProbs, self.
                        responseProbs[indexes])
                return responseProbs

    def testResponseProb(self, probBound, responseProbs=None):
        """
        Checks if the proportions calculated by the method
            getResponseProb are
        greater than n to decide if there's a response or not for
            every bin in the experiment
        :probBound: probability bound that the response probability
            must exceed to be considered as a response
        :return: int[] array of true or false values that represents
            wether there's a response
        for that bin
        """
        if(responseProbs is not None):
            response = []
            for interValues in responseProbs:
                if(np.where(interValues >= probBound)[0].shape[0] >
                    0):
                    response.append(1.0)
                else:
                    response.append(0.0)
            return response
        else:
            return (self.responseProbs >= probBound).astype(float)


    def calculateBinsResponseProb(self, baselineIntervals):
        """
        For each bin in the experiment, calculates its response
            probability. In order to do so,
        baseline probability distribution is calculated, excluding
            the testing bin if its belongs
        to baseline. Then each bin probability distribution is test
            against the baseline one, getting
        the lower bound of the response probability
        :baselineIntervals: float[float[]] array of arrays containing
            the intervals taken as baseline
        :return: float[] intervals start points of bins
        :return: float[] an array containing the probability response
            for each bin in the experiment
        """
        baselineBins = np.array([])
```

```
        self.responseProbs = np.array([])

        # gets the indexes of the bins inside the baseline intervals
        for interval in baselineIntervals:
            baselineBins = np.append(baselineBins, np.where(np.
                logical_and(self.intervals >= interval[0], self.
                intervals < interval[1])))
        baselineBins = np.unique(baselineBins)

        # for each bin, gets its response prob
        for i,interval in enumerate(self.intervals):
            # excludes the tested bin before calculating the baseline
                prob distribution
            bins = baselineBins[np.where(baselineBins != i)]
            # calculates the total baseline prob distribution
            baselineProb = np.add.reduce(self.binProbs[bins.astype(
                int), :], axis=0)/baselineBins.shape[0]
            testedBinProb = self.binProbs[i]
            testedBinSpikes = self.spikeCount[i]
            # gets the response probability for bin and stores it
            self.responseProbs = np.append(self.responseProbs, self.
                independentResponseProb(baselineProb, testedBinProb,
                testedBinSpikes.astype(int)))
        return self.intervals, self.responseProbs


    def calculateBinDistribution(self, totalTime, maxSpikes=20):
        """
        For each bin within the experiment, calculates the spikes
            probability distribution
        :totalTime: float total time of experiment
        :maxSpikes: int maximum number of spikes per bin. Set to 20
            by default
        :return: void. Stores internally the calculated values
        """
        # calculates start points of bins
        intervalsStartPoint = np.arange(0, totalTime, step=self.step)
        self.intervals = intervalsStartPoint[np.where(
            intervalsStartPoint + self.binSize <= totalTime)]

        data = self.experiment.data
        nTrials = self.experiment.nTrials

        self.binProbs = np.ndarray((self.intervals.shape[0],
            maxSpikes))
        self.spikeCount = np.ndarray((self.intervals.shape[0],
            nTrials))

        for i,start in enumerate(self.intervals):
            # gets the spikes (events) inside a bin
            events = data[np.where(np.logical_and(data[:,1] >= start,
```

```
                data[:,1] < start + self.binSize))]
            events[:, 0] = events[:, 0] - 1

            # count the number of spikes for each trial in the
                experiment
            spikeCount = np.bincount(events[:,0].astype(int))


            # if during some trials there has been no spike, the
                array is fill with zeros
            if(spikeCount.shape[0] < nTrials):
                spikeCount = np.append(spikeCount, np.zeros(nTrials -
                    spikeCount.shape[0]))

            # divides the total spike counts by the number of trials
                to get the probability of n spikes
            probs = np.bincount(spikeCount.astype(int))/float(nTrials
                )

            # fills array with zeros until having maxSpikes values
            if(probs.shape[0] < maxSpikes):
                probs = np.append(probs, np.zeros(maxSpikes - probs.
                    shape[0]))
            else:
                print("ERROR: max spike count reached. You should
                    increment maxSpikes parameter.")
                return

            # stores the spike counts and the probs for each bin
            self.binProbs[i, :] = probs
            self.spikeCount[i, :] = spikeCount

    def getResponseProb(self, stimuliIntervals, baselineIntervals):
        stimuliIndexes, stimuliResponseProbs = self.
            calculateProbDistribution(stimuliIntervals)
        return stimuliResponseProbs


    def calculateProbDistribution(self, stimuliIntervals):
        """
        Calculates the probability distribution for several
            stimuliIntervals
        :stimuliIntervals: intervals that are going to be tested for
            responseProbs
        :return: the indexes of bins inside the intervals specified,
            the reponse probability for each interval
        """

        responseProbs = []
        stimuliIndexes = []
        for interval in stimuliIntervals:
```

```
            stimuliIndex = np.where(np.logical_and(self.intervals >=
                interval[0], self.intervals + self.binSize <= interval
                [1]))
            stimuliIndex = np.asarray(stimuliIndex)
            stimuliIndexes.append(stimuliIndex[0])

            responseProbs.append(np.mean(self.responseProbs[
                stimuliIndex[0]]))
        return stimuliIndexes, responseProbs


    def independentResponseProb(self, baselineProbs, stimuliProbs,
        spikes):
        """ gets lower bound of response probability as P(s1, s2...,
            sn | baseline)/P(s1, s2,.., sn | odor)
        # priores are not necessary since we know they're between 0
            and 1
        :baselineProbs: baseline probability distribution
        :stimuliProbs: stimation window probability distribution
        :spikes: series of spikes in the interval being tested
        :return: float the low bound of the response probability
        """
        val = np.multiply.reduce(baselineProbs[spikes])/np.multiply.
            reduce(stimuliProbs[spikes])
        if(val > 1):
            val = 1
        return 1 - val
```

B.3.7. BayesianTestBootstrap

```
import BayesianTest
import numpy as np

class BootstrapBayesianMethod(BayesianTest.RawBayesianMethod):
    def __init__(self, binSize, step, experiment, nSamples,
        sampleSize):
        """
        Constructor of the class
        :binSize: float binSize
        :step: float distance between bin start points
        :experiment: experiment to be testedBinProb
        """
        super(BootstrapBayesianMethod, self).__init__(binSize, step,
            experiment)
        self.nSamples = nSamples
        self.sampleSize = sampleSize

    def calculateBinDistribution(self, totalTime, maxSpikes=20):
        """
        For each bin within the experiment, calculates the spikes
            probability distribution
        using nSamples bootstrap samples of size sampleSize
```

```
        :totalTime: float total time of experiment
        :nSamples: number of bootstrap samples
        :sampleSize: size of bootstrap sample
        :maxSpikes: int maximum number of spikes per bin. Set to 20
            by default
        :return: void. Stores internally the calculated values
        """
        # calculates start points of bins
        intervalsStartPoint = np.arange(0, totalTime, step=self.step)
        self.intervals = intervalsStartPoint[np.where(
            intervalsStartPoint + self.binSize <= totalTime)]

        data = self.experiment.data
        nTrials = self.experiment.nTrials

        self.binProbs = np.ndarray((self.intervals.shape[0],
            maxSpikes))
        self.spikeCount = np.ndarray((self.intervals.shape[0],
            nTrials))

        bootstrapSamples = np.ndarray((self.nSamples, self.sampleSize
            ))
        for i,start in enumerate(self.intervals):
            # gets the spikes (events) inside a bin
            events = data[np.where(np.logical_and(data[:,1] >= start,
                data[:,1] < start + self.binSize))]
            events[:, 0] = events[:, 0] - 1

            # count the number of spikes for each trial in the
                experiment
            spikeCount = np.bincount(events[:,0].astype(int))

            # if during some trials there has been no spike, the
                array is fill with zeros
            if(spikeCount.shape[0] < nTrials):
                spikeCount = np.append(spikeCount, np.zeros(nTrials -
                    spikeCount.shape[0]))

            indexes = np.random.uniform(0, spikeCount.shape[0], size
                =(self.nSamples, self.sampleSize))
            bootstrapSamples = spikeCount[indexes.astype(int)]
            bootstrapSamples = bootstrapSamples.astype(int)
            bootstrapSamples = [np.bincount(sample)/float(nTrials)
                for sample in bootstrapSamples]
            bootstrapSamples = [np.append(probs, np.zeros(maxSpikes -
                probs.shape[0])) for probs in bootstrapSamples if
                probs.shape[0] < maxSpikes]
            finalProbs = np.ndarray((self.nSamples, maxSpikes))
            finalProbs[:, :] = bootstrapSamples
            probs = np.mean(finalProbs, axis=0)
```

```
            # stores the spike counts and the probs for each bin
            self.binProbs[i, :] = probs
            self.spikeCount[i, :] = spikeCount
```

B.3.8. GaussianMethod

```
class GaussianMethod(TestResponseMethod):
    """
    This class implements the Gaussian test used in Orive et al. to
        detect neural
    responses. This method works by calculating the standard
        deviation of the sample. If an interval exceeds
    n times the SD of the mean firing rate during baseline at least
        in one bin, it's considered a response.
    To choose the bin size that best fits the mean firing rate, a
        cost function is calculate for each bin size
    and the one that minimizes the cost is chosen as best bin size.
        This method for finding the best bin size
    is proposed in []
    """
    def __init__(self, experiment, totalTime, step, maxWidth):
        """
        Constructor of the class
        :experiment: Experiment experiment to be testedBinProb
        :totalTime: float experiment total time
        :step: float width of bin sizes
        :maxWidth: float maximum bin size
        """
        self.experiment = experiment
        self.totalTime = totalTime
        self.bestBinSize, self.costs = self.getBestBinSize(experiment
            , totalTime, step, maxWidth)


    def getResponseProbs(self, intervals=None):
        return self.kStds

    def calculateBinDistribution(self, step=None, binSize=None):
        """
        Divides the experiment time into bins of binSize seconds and
            counts the
        total number of spikes across all trials for the bin. Finds
            the mean firing rate
        by dividing the spike count by the number of trials
        :step: float step between the start points of bins. Already
            inicialized to the value
        specified in the constructor or to None if no value was given
            .
        :binSize: size of the bins. Already inicialized to the value
            specified in the constructor
        or to None if no value was given.
        :return: float[] start points of bins
```

```
        :return: float[] spike count for each bins
        :return: float[] mean firing rate for each bin
        """
        if(step == None):
            step = self.step
        self.binSize = binSize
        if(self.binSize == None):
            self.binSize = self.bestBinSize

        intervalsStartPoint = np.arange(0, self.totalTime, step=step)
        self.intervals = intervalsStartPoint[np.where(
           intervalsStartPoint + self.binSize <= self.totalTime)]

        data = self.experiment.data
        nTrials = self.experiment.nTrials

        origin = 0
        k = np.array([])

        for start in self.intervals:
            k = np.append(k, np.where(np.logical_and(data[:,1] >=
               start, data[:,1] < start + self.binSize))[0].size)

        self.meanFiringRate = k/nTrials
        return self.intervals, k, self.meanFiringRate


    def costFunction(self, mean, variance, nTrials, binSize):
        """
        Costs function to find the bin size that minimizes it.
        :mean: float mean firing rate for certain bin size
        :variance: float variance for a certain bin size
        :nTrials: int number of trials in experiment
        :binSize: float size of the bin
        :return: float the cost function value for the parameters
           specified
        """
        return (2*mean - variance)/pow(nTrials*binSize, 2)


    def getBestBinSize(self, experiment, totalTime, step, max):
        """
        Applying the method proposed in [], finds the bin size that
           best fits
        the underlying mean firing rate distribution during the
           experiment
        :experiment: Experiment experiment that's being analyzed
        :totalTime: float duration of experiment
        :step: float step between bins start points
        :max: float maximum bin size to test
        :return: float best bin size
```

```
        :return: float[] costs array
        """
        costs = np.ndarray((floor(max/step), 2))
        delta = step
        counter = 0

        # While the maximum bin size hasn't been reached
        while(delta <= max):
            #calculates the spike count for each bin
            _, k, _ = self.calculateBinDistribution(delta, delta)
            # calculates the mean of the spike counts
            mean = np.mean(k)
            # calculates the variance of the spike counts
            variance = np.var(k)
            # gets value of cost function for this bin size
            cost = self.costFunction(mean, variance, experiment.
                nTrials, delta)
            # stores the cost
            costs[counter, :] = [delta, cost]
            # next bin size
            delta += step
            counter += 1

        costs[:, 1] = costs[:, 1]/costs[:, 0]
        # chooses the bin size that minimizes the cost function
        return costs[np.argmin(costs[:, 1]), 0], costs


    def testResponseN(self, n, kStds=None):
        """
        Checks if the proportions calculated by the method
            getResponseProb are
        greater than n to decide if there's a response or not for
            every bin in the experiment
        :n: int number of times that the stimulation mean firing rate
            must exceed
        the std of the mean firing rate during baseline
        :return: boolean[] array of true or false values that
            represents wether there's a response
        for that bin
        """
        if(kStds is not None):
            response = []
            for interValues in kStds:
                if(np.where(interValues >= n)[0].shape[0] > 0):
                    response.append(1.0)
                else:
                    response.append(0.0)
            return response
        else:
            return (self.kStds >= n).astype(float)
```

```
    def calculateBinsResponseProb(self, baselineIntervals):
        """
        Calculates the response probability for each bin within the
            experiment.
        In order to do so, every bin is compared against the bins in
            the baselineIntervals.
        If the bin being tested is include in the baselineIntervals,
            it is removed from baseline
        :baselineIntervals: array[float[]] baseline intervals in the
            experiment
        """

        baselineBins = np.array([])
        self.kStds = np.array([])
        # gets the indexes of the bins inside the baseline intervals
        for interval in baselineIntervals:
            baselineBins = np.append(baselineBins, np.where(np.
                logical_or(self.intervals >= interval[0], self.
                intervals < interval[1])))
        baselineBins = np.unique(baselineBins)

        # for each bin, gets the proportion between its mean firing
            rate and the baseline std
        for i,interval in enumerate(self.intervals):
            # excludes the tested bin before calculating the baseline
                 prob distribution
            bins = baselineBins[np.where(baselineBins != i)]
            baselineStd = np.std(self.meanFiringRate[bins.astype(int)
                ])
            self.kStds = np.append(self.kStds, self.meanFiringRate[i
                ]/baselineStd)



    def getResponseProb(self, stimuliIntervals, baselineIntervals):
        """
        Applying the Gaussian test, finds how many times the mean
            firing rate during
        the stimulation period is greater than the mean firing rate
            during baselineIndex
        :stimuliIntervals: array[array] array of arrays containing
            the stimulation intervals.
        Example of format: [[4,5], [6,8]]
        :baselineIntervals: array[array] array of arrays containing
            the baseline periods. The
        format is the same that for stimuliIntervals
        :return: float[] kStds proportion between stimuli mean firing
             rate and baseline sd
        :return: float stimuliFiringRateMean mean firing reate during
```

```
        stimulation periods
    :return: float baselineStd standard desviation of mean firing
        rate during baseline periods
    """
    baselineIndex = np.array([])
    # gets the indexes of the bins inside baseline intervals
    for interval in baselineIntervals:
        baselineIndex = np.append(baselineIndex, np.where(np.
            logical_and(self.intervals >= interval[0], self.
            intervals + self.binSize <= interval[1])))
    # gets mean firing rate for each bin inside baseline
        intervals
    baselineMeans = self.meanFiringRate[baselineIndex.astype(int)
        ]
    # gets mean firing rate for the whole baseline interval
    baselineMean = np.mean(baselineMeans)
    # gets the standard deviation for the whole baseline interval
    baselineStd = np.std(baselineMeans)

    stimuliFiringRateMean = []
    kStds = []
    # gets the indexes of the bins inside stimulation intervals
    for interval in stimuliIntervals:
        stimuliIndex = np.where(np.logical_and(self.intervals >=
            interval[0], self.intervals + self.binSize <= interval
            [1]))
        stimuliIndex = np.asarray(stimuliIndex)
        # gets mean firing rates for bins inside the stimulation
            interval
        meanFiringRates = self.meanFiringRate[stimuliIndex[0].
            astype(int)]
        # gets the proportion between the mean firing rate during
             stimulation and the standar deviation of baseline
        kStds.append(meanFiringRates/baselineStd)
        # stores the proportion
        stimuliFiringRateMean.append(self.meanFiringRate[
            stimuliIndex[0].astype(int)])

    return kStds, stimuliFiringRateMean, baselineStd
```

B.3.9. Plotter

```
"""
This module contains functions to plot several representations of
data
"""


def plotBinsResponseProb(responseProbs, intervals, log=True):
    """
    Prints a graphic representing the response probability of the
```

```
        neuron vs. time
    for an experiment.
    :responseProbs: float[] 1D array containing the response
        probabilities
    :intervals: float[] 1D array containing the start point of
        intervals
    :log: boolean if set to True, the response probability is printed
         in logarithmic
    scale. Set to True by default
    :return: Figure two pyplot figures with the representation of the
         data
    """
    plt.figure
    f = plt.figure()
    plt.title("Response Probabilty")
    plt.xlabel("Time (s)")
    plt.ylabel("Response Probability")
    if log:
        responseProbs = 1-np.log(responseProbs)
    plt.plot(intervals, responseProbs)
    return f


def plotRaster(data, experimentSet, title, nTrials, trialTime,
   stimuliTimes):
    """
    Plots the spike raster. Y axis represents number of trials, while
         X axis representations
    time. Vertical lines are printed delimiting the stimulation
        period
    :data: float[][] 2D array containg the experiment data (number
        of trial and time of spike)
    :experimentSet: experimentSet containing the experiment to be
        plotted
    :title: title of the plot
    :nTrials: number of trials in the experimentSet
    :trialTime: time per trialTime
    :stimuliTimes: times at which the system is stimulated
    :return: Figure two pyplot figures with the representation of the
         data
    """

    f = plt.figure()
    plt.plot(data[:, 1], data[:, 0], "r|", mew=2.0)


    for time in stimuliTimes:
        plt.plot((time, time), (0, nTrials), 'k-')
        plt.plot((time + 1, time + 1), (0, nTrials), 'k-')
    plt.xlabel("Tiempo(s)")
    plt.ylabel("Prueba")
```

```
    plt.title(title)
    plt.axis(v=[0, trialTime, 1, trialTime])
    plt.xlim(0,trialTime)
    plt.ylim(0,trialTime)
    return f

def plotSurface(x, y, z):
    """
    Plots a surface showing the evolution of response probability
       during an experiment
    depending on testing window width and start point
    :x: float[] 1D array containing window start point values
    :y: float[] 1D array containing window width values
    :return: Figure two pyplot figures with the representation of the
        data
    """
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.set_xlabel('Window start')
    ax.set_ylabel('Window width')
    ax.set_zlabel('Probability')
    ax.text2D(0.05, 0.95, "Response probability lower bound",
       transform=ax.transAxes)
    xgrid = np.linspace(x.min(), x.max(), x.shape[0])
    ygrid = np.linspace(y.min(), y.max(), y.shape[0])
    xgrid, ygrid = np.meshgrid(xgrid, ygrid)

    surf = ax.plot_surface(xgrid, ygrid, z, cmap=cm.coolwarm,
       linewidth=0, antialiased=False, rstride=1, cstride=1)
    ax.set_zlim(0, 1.10)
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.zaxis.set_major_formatter(FormatStrFormatter('%.2f'))



def plotFalsePositives(title, bounds, fraction, nFalses):
    """
    Prints number of false positives against the p_r selected and
       Prints
    the p_r s as vertical bars within the plot
    :bounds: different p_r tested to be displayed as vertical bars
    :fraction: percentage corresponding to each p_r in bounds
    :nFalses: matrix with the p_r tested and their number or false
       positives
    """
    f = plotXY(title, "Probability Bound", "N false positives",
       nFalses[:, 0], nFalses[:, 1])

    for i,bound in enumerate(bounds):
        plt.plot((bound, bound), (0,1.1), 'k-')
        plt.annotate("P_" + str(fraction[i]), [bound, np.random.rand
```

```
            ()])

    return f

def plotXY(title, xlabel, ylabel, x, y):
    """
    Plots a generic Y vs. X graphic
    :title: String title of the graphic
    :xlabel: String X axis label
    :ylabel: String Y axis label
    :x: float[] 1D array containing values for X
    :y: float[] 1D array containing values for Y
    :return: Figure two pyplot figures with the representation of the
        data
    """
    f = plt.figure()
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    axes = plt.gca()
    axes.set_ylim([0, 1.1])
    plt.plot(x, y, 'o-', linewidth=4)

    return f


def fwhmSigma(fwhm):
    return fwhm / np.sqrt(8 * np.log(2))

def getSmoothGaussianVal(sigma, xPos, vals):
    """
    Smoothed a value by using a gaussian
    :sigma: gaussian sigma
    :xPos: position of the value to smooth
    :values: val to Smooth
    :return: new smoothed value
    """
    gaussian = norm(loc=xPos, scale=sigma)
    gaussianVals = gaussian.pdf(range(vals.shape[0]))
    #print gaussianVals
    gaussianVals = gaussianVals / sum(gaussianVals)
    newVals = gaussianVals * vals
    newVal = sum(newVals)
    return newVal


def plotSensitivity(nOlors, probs, probBound, sigma):
    """
    Plots the sensitivity of the system with bars and Smoothed
    by a gaussian
    :nOlors: number of stimuli in the experiment
```

```
    :probs: response probabilities
    :probBound: p_r chosen
    :sigma: sigma of the Gaussian used to smooth
    """
    f = plt.figure()
    plt.bar(nOlors, probs, width=1.0, color=(1.0, 0.69, 0.4))
    smoothProbs = np.array([])
    for i in range(probs.shape[0]):
        smoothProbs = np.append(smoothProbs, getSmoothGaussianVal(
            sigma, i, probs))

    xSmooth = np.linspace((nOlors + 0.5).min(), (nOlors + 0.5).max(),
        300)
    ySmooth = spline(nOlors + 0.5, smoothProbs, xSmooth)
    plt.plot(xSmooth, ySmooth, 'o-', linewidth=3, color='#FF4704')
    axes = plt.gca()
    axes.set_ylim([0, plt.ylim()[1]])
    plt.xlabel("# olors")
    plt.ylabel("Sensitiviy")
    plt.title("Sensitivity for " + str(probBound))
    plt.show()


def plotPSTHD(intervals, meanFiringRate, spikeCount, binSize):
    """
    Plots the PSTHD of an experiment. Returns two graphics: the first
        one
    shows time vs. #of spikes; the second one shows time vs. mean
       firing rate
    :intervals: float[][] 2D array with the histogram intervals
    :meanFiringRate: float[] mean firing rate value for each bin
    :spikeCount: float[] total spike count for each binSize
    :binSize: size of bin
    :return: Figure two pyplot figures with the representation of the
        data
    """
    print intervals
    f1 = plt.figure()
    plt.bar(intervals, meanFiringRate, width=binSize, color
       =(0.2588,0.4433,1.0))
    plt.title("PSTH (mean)")
    plt.xlabel("Time (s)")
    plt.ylabel("Mean firing rate")

    f2 = plt.figure()
    plt.bar(intervals, spikeCount, width=binSize, color
       =(0.2588,0.4433,1.0))
    plt.title("PSTH")
    plt.xlabel("Time (s)")
    plt.ylabel("# Spikes")
```

```
    return f1, f2


def plotCostFunction(cost, title):
    """
    Plots the cost function for the error commited by the binSize
       with respect
    to the stimation of the mean firing rate
    :cost: float[][] 2D array with value of Y for each X
    :return: Figure two pyplot figures with the representation of the
        data
    """
    f = plt.figure()
    plt.title(title)
    plt.xlabel("Bin Size")
    plt.ylabel("C(binSize)")
    plt.plot(cost[:, 0], cost[:, 1], mew=3.5)
    return f


def activationPlot(data, title, xlabel, ylabel):
    """
    Plots a black/white activation plot of data first dimension
    columns and data second dimension rows. Black corresponds to the
       response
    and white to its absence
    :data: matrix with 0/1 values for each combination of sytem and
       stimuli
    :title: title of the plot
    :xlabel: x axes label
    :ylabel: y axes label
    """
    f = plt.pcolor(data, cmap='gray_r')
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    axes = plt.gca()
    axes.set_xlim([0, data.shape[1]])
    return f

# IDEA: can show() receive a figure/figures as argument and display
   just those?
def show():
    """
    Shows all the figures creating during a session
    """
    plt.show()
```